Sound Design

An Artificial Intelligence Approach

Eduardo Reck Miranda

Submitted for the Degree of Doctor of Philosophy

University of Edinburgh

1994

Declaration:

I declare that this thesis has been composed by myself and that it is the result of my own research work.

Eduardo Reck Miranda
Edinburgh, October 20, 1994.

Acknowledgements

I thank to my supervisors Peter Nelson and Alan Smaill, for their support, advise and input on repeated rewritings of this thesis. My thanks to Gerhard Eckel and Gerard Assayag (Ircam); Jean-Gabriel Ganascia and Jean Pachet (University of Paris); Tim Anderson and David Rossiter (University of York); Nigel Osborne (Edinburgh University) for useful discussions and critics. Thanks to composers Stephen Davismoon, Paul Keenan, Peter Nelson and Pierre-Yves Rolland, for the time they spent experimenting with the system and for their useful comments. Finally, thank you to Alexandra Jones, who helped with typing and proof reading.

Abstract

In this thesis we report how we have attempted to devise an Intelligent System for Sound Design (ISSD) that (a) allows the composer to design sounds by communicating with the computer by means of his or her own vocabulary of sound descriptors (by using words in English, for example) rather than in terms of numerical settings and (b) provides and intelligent exploratory aid, so that the computer can work in co-operation with the user by offering tools for exploration of the possibilities of a synthesis algorithm.

Modern computer technology enables the production of a virtually limitless variety of sounds by providing substantial access to the parameter settings of synthesis algorithms. However, the design of sounds using a synthesis algorithm is still accomplished in a very old-fashioned way: by feeding the algorithm with streams of numerical data. Furthermore, these numbers are usually worked out manually. For example, a composer who works with the Csound synthesis programming language must master Csound for implementing a synthesis algorithm and also must specify all the input parameter values for the production of every single sound. Depending on the complexity of the algorithm, there might be cases where over a hundred parameters need to be specified for each sound event. In such a situation the imagination of the composer can easily become vulnerable to time-consuming, non-musical tasks.

We argue that the power of the computer could also provide better ways for the composer to express his requests to the synthesis algorithm at hand and moreover, provide appropriate aid for the exploration of sonic ideas.

To this end, we propose an Artificial Intelligence (AI) approach to sound design systems, which focuses on sound design as a knowledge-based kind of intelligent behaviour. We consider that sound design involves the explicit organisation, application and generation of knowledge. AI is aimed here at helping the composer to handle this knowledge by means of suitable knowledge representation and machine learning techniques.

We carried out this investigation through the design of a case study ISSD. Although our case study implements a particular synthesis technique (namely, subtractive synthesis of formants), for which we defined a particular vocabulary of sound descriptors, our aim is to propose an approach for the design of an ISSD. We explain and discuss the underlying philosophy of our approach by illustrating how we designed the case study system.

Contents

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 1

Introduction

Recent studies in acoustics, psychoacoustics, the psychology of music and cognitive musicology have vastly expanded our knowledge of the nature and perception of sounds and music. The sound domain of Western music is no longer demarcated by the boundaries of traditional acoustic instruments. Nowadays, composers have the opportunity to create music with an infinite variety of sounds, ranging from "natural sounds" (those produced by acoustic devices and different sorts of mechanical excitation; such as the sounds produced by blowing a pipe (Rossing, 1990)) to synthesised, "artificial sounds" (those sounds that cannot be produced by acoustic devices; such as the sounds produced by a cellular automata-based granular synthesiser (Miranda et al., 1992; Miranda, 1995)).

Quoting Max Mathews and John Pierce (1987, pp. 90), *"the electronic instruments and computers provide a virtually limitless universe of sounds through which composers and performers can express their thoughts and feelings."* Computer technology offers the most detailed control of the internal parameters of synthesised sounds, which enables composers to become more ambitious in their quest for a more effective use of computer synthesis technology. In this case however, the task of sound composition becomes more complex. A composer can set the parameters for the production of an immeasurable variety of sounds, but this task is still accomplished unnaturally by inputting streams of numerical data (as in the case of the Csound score files, for example (Vercoe, 1991)). Even if the composer knows the role played by each single parameter for synthesising a sound, it is both very difficult and tedious to ascertain which values will synthesise the sound he wants to produce. Moreover, the composer often needs to master a sound synthesis programming language in order to communicate with the computer (as in the case of CLM, for example (Schottstaedt, 1992; 1994)). Even if he masters this language, the design of an instrument is not a straightforward task. In such a situation, higher processes of inventive creativity and abstraction become subsidiary to time consuming, non-musical tasks. Composers need a better working environment.

It seems that the interdisciplinary knowledge we have about the nature and perception of sounds (that is, acoustics, psychoacoustics, psychology of music, etc.) has not been taken into account by sound synthesis software engineers. Better sound design systems can be provided if we devise ways for including this knowledge in a sound design software engineering methodology. This situation can be improved by combining computer sound synthesis technology with Artificial Intelligence (AI) techniques in sound design systems. AI techniques are aimed here to help us to devise sound synthesis systems that take into account the interdisciplinary knowledge mentioned above.

Producing a desired sound on a musical instrument such as a clarinet, for example, depends fundamentally upon sub-cognitive physical skills - to operate keys, control air-flow rate, pressure profiles, etc. With training and practice, musicians develop an "inner ear" with which they mentally "hear" a sound. They also develop the fine physical control required to generate and control the desired sounds on their particular instrument. This process is, however, not typically cognitively accessible and is therefore difficult to make explicit.

A similar situation exists for the relationship between an imagined sound, a real sound and a sound description. If, for example, we ask our clarinettist to play a melancholy sound, he would have no difficulty imagining a suitable sound and producing one that satisfies our request; however, the clarinettist would find it difficult to explain how this effect is produced.

If instead, our clarinettist turns to computer sound synthesis, this lack of an explicit understanding of how imagined sounds are produced and described presents significant problems. The computer synthesis of sounds is fundamentally controlled, not by sub-cognitive physical skills, but by setting values in algorithms which control digital sound synthesis devices (oscillators and filters, for example). The task of sound design using the techniques of computer sound synthesis is therefore qualitatively different from sound design using acoustic instruments. It is more complex in terms of the cognitive problem solving skills required and correspondingly less complex in terms of the sub-cognitive physical skills required.

When designing computer synthesised sounds to be used in a piece of music (for example, a piece of electroacoustic music), composers seem to have an intuition about the possibilities of how these sounds could be organised into a musical structure. In

order to design these sounds, composers often explore a variety of possible solutions by trying out possibilities within a certain personal style or idiom. It is therefore desirable to provide a sound design system that helps the user to express his ideas to the computer, as well as to explore various alternatives during the design process.

In this thesis we propose an approach for combining sound synthesis with AI. In order to test this approach we have designed a case study AI sound design system that allows the design of sounds by thinking in terms of qualitative descriptions (for example, by using words in English) rather than in terms of numerical streams. Moreover, this system also works in co-operation with the user by providing support for the exploration of ideas.

It is worth emphasising that we do not aim at a system tied either to a specific synthesis algorithm or to a specific vocabulary for sound description. Rather, we aim at a software that allows user-configuration according to the synthesis algorithm he wishes to use and according to a personal vocabulary for sound description.

Although the system we propose in this thesis uses a specific sound synthesis method and a specific vocabulary for sound description, during its design process we addressed several issues that give us various insights towards a more generic sound design software. Thus, in this thesis we present an approach to sound design systems based upon the implementation of a case study. In general terms, our proposed approach to sound design is aimed at a system that:

(a) allows the user to specify a synthesis algorithm and then to represent it in such a way that he can coherently create his own labels (that is, a vocabulary of sound descriptors) to refer to the sounds produced by this synthesis algorithm, to the attributes that describe these sounds and also to the synthesis parameters themselves

(b) allows the user to explore the capabilities of a synthesis algorithm by inputting requests that use these user-defined labels

(c) provides mechanisms aimed at supporting the user in the exploration of the capabilities of a synthesis algorithm

(d) is able to acquire new qualitative information about sounds (that is, augment its knowledge about the vocabulary) in the user's own terms.

Of course there are some constraints that should be carefully observed in order to work effectively with this approach. The two most crucial constraints we can indicate at this point are as follows:

(a) it is up to the user to design suitable instruments, choose good labels and keep track of them coherently in the system

(b) there will probably be certain synthesis models that better suit our approach to sound design than others.

In order to address all these issues, we have to study the relationship between a mentally conceived sound, its timbre characteristics and its symbolic representation and manipulation of this representation.

The methodology of our investigation approaches the problem at three levels. Using AI terminology, these levels are: the *knowledge level*, the *symbolic level*, and the *engineering level.*

At the knowledge level we attempt to identify what a system has to "know" in order to synthesise a sound from its attributional, qualitative description. In AI jargon, we say that at this level we ought to identify the *body of knowledge* required to accomplish a *goal.* The main problem at this level is mapping a qualitative sound description to its respective sound synthesis parameters; we attempt here to devise a solution for this problem. At the symbolic level, we propose a system architecture which embodies the knowledge level; we define data structures that integrate and organise the information of the body of knowledge, plus the algorithms that manipulate this information in order to accomplish the goals of the system. At the next stage, the engineering level, we study the implementation issues of the proposed architecture.

The proposed architecture has both built-in and open-ended modules. The open-ended modules are user specified. In these modules the user specifies the information of the body of knowledge, whereas the built-in modules provide the algorithms that compute this information.

During the design process of the case study system, we considered the type of composer who has some experience of a synthesis programming language, such as Csound, but finds designing sound with it difficult and tedious.

Our case study system provides two levels of operation: the *sound design level* and the *instrument design level.* At the sound design level, the system provides a series of commands and other facilities for the design of sounds with the provided synthesis algorithm (that is, the instrument) and vocabulary. At the instrument design level, the user customises the system, so that he implements a synthesis algorithm and specifies a vocabulary.

For the sound design level, the user is required to know:

(a) the commands for sound design

(b) the capabilities of the provided instrument

(c) the provided vocabulary.

Conversely, for the instrument design level the user is required to know how to customise the system. We suggest a method for doing this, which we achieved through our AI approach to sound design. This assumes that the vocabulary for sound description is intimately related to the way in which the instrument is designed. If the user wishes to refer to the vibrato quality of sounds, for example, then the vibrato generator must be available. One should not ask the system to produce a wobble sound if the instrument is not able to make *glissandi*, for example.

It is important to note that at this stage we do not deal with the development of a language for implementing synthesis algorithms at their signal processing level. This can be done by means of any available sound synthesis language (dealt with in Chapter 7). For this case study system, the implementation of the instrument is done in Csound.

We begin the thesis by introducing our view of sound synthesis as a design problem. We then indicate the capabilities we think to be desirable in an Intelligent System for Sound Design (ISSD) (Chapter 2). In Chapter 3, we assess four examples of related research work that in some way have inspired our own research. Then in Chapter 4,

we introduce the reader to some background concepts needed to understand the underlying ideas of this work. Bearing in mind the aforementioned methodology of investigation, we then study the design and implementation of the case study system (Chapters 5, 6, and 7). Finally, in Chapter 8, we

(a) attempt to evaluate the case study system

(b) review our approach to ISSD design

(c) present some conclusions

(d) envisage further work.

Although the evaluation of the case study system focuses mainly upon the higher level (that is, the sound design level) of operation, its results also address the method that we suggested for the customisation of the instrument and vocabulary at the lower level. Other information, such as the detailed description of the synthesis method used here, is given in various appendices at the end of the thesis.

To summarise, in this thesis we propose a completely new approach to improve current sound synthesis systems. Sound design using computers is a complex task because it requires the efficient management of a large amount of knowledge. Current sound synthesis systems do not use the power of the computer to assist the sound designer to manage this knowledge. The design process using these systems is time-consuming and engages the user in a considerable amount of non-musical tasks. For example, he must feed the system with streams of numerical synthesis parameters for each single sound. Moreover, these values are often worked out manually. Our approach aims to alleviate this problem by using AI; the sound designer can therefore delegate certain tasks to the computer, thus allowing him more time for more creative aspects. In addition, we use AI to devise a higher level interface to enable the user to communicate with the system by using his own vocabulary for sound description, in English: for example, *fast vibrato and low register sound.* We use knowledge representation and machine learning techniques in order to provide the computer with knowledge about sound synthesis and sound description which can be formalised. In order to test our approach we devised a prototype system which responds to a user-customised vocabulary of sound descriptors in English and assists the user during the design process.

Chapter 2

2 Sound Synthesis and Design

In this chapter, we examine how sound synthesis can be studied as a design problem and indicate the desirable capabilities we believe to be important in an intelligent sound design system.

We begin by explaining how we view design as knowledge-based intelligent behaviour and distinguish two approaches to design systems. Next, we discuss sound synthesis as design and point to some features we believe to be desirable in an ISSD. We end the chapter with a brief discussion of evaluation issues.

2.1 Understanding design

2.1.1 Design as knowledge-based intelligent behaviour

Design is a complex kind of intelligent behaviour. It is concerned with engaging in cognitive and physical acts in order to establish the suitability and effectiveness of our creations prior to actually constructing them. In attempting to solve design problems, designers explore the space of possible solutions by trying out possibilities and investigating their consequences (Logan et al., 1992).

One cannot hope to fully understand design by adopting a single perspective on its study, but we must combine the perspectives of many different disciplines. Nevertheless, we are interested for present purposes, in a limited aspect of design: *design as an explicitly knowledge-based kind of intelligent behaviour.* Thus we assume that it involves the explicit organisation, application and generation of knowledge (Smithers et al., 1989).

Artificial Intelligence is a science which aims at understanding intelligent behaviour and how it might be artificially created to serve specific goals (Luger and Stubblefield, 1989; Haton and Haton, 1989). In this context, in order to understand design as a

kind of intelligent behaviour we need ways to describe and express aspects of the behaviour being investigated: how we think this behaviour can be modelled and how we think it can possibly be aided, or even simulated by a computer. These three needs suggest three different levels of study: the *knowledge level,* the *symbolic level,* and the *engineering level* (Newell, 1981). We have adopted these levels in carrying out this research.

2.1.2 Focusing on an approach to design systems

We distinguish two approaches to design systems. One is to build computer programs which replicate human design behaviour, or which simply create design artefacts with no human intervention. The other is to try to build systems which provide support for the designers. Although these can be complementary, our research focuses on the second approach. In so doing, we believe that we reduce our demand for a complete formalisation of the problem, that is, not all the reasoning and problem solving have to be done by the computer, but only certain aspects of it. We do not intend the computer to replace human subjectivity, but rather to support it.

We believe that by adopting the second approach we might be able to produce a sound design system which can aid in structuring and in making explicit the knowledge of sounds, by providing mechanisms for enhancing the process of engendering new information. Moreover, this approach offers one feature believed to be very important in design systems, namely *feedback.*

Feedback enables a two-way communication between the user and the computer. It allows the exchange of actions between the user and the system, so that the user transforms the system and the system transforms the user (Piaget, 1947). In this case, an intelligent design support system is expected to interact with the user by offering alternative paths to achieve a solution which perhaps the user had never thought of before (Edmonds, 1993). Many creative insights may arise as a result of the interaction. Feedback allows the user to build up experimental solutions, examine them and modify them at will. Analysis of the results is then fed back into an improved solution. Successive iterations ought to result not only in a better solution, but should also allow better insight into the problem space which it encompasses. The computer is seen here as an instrument for the gradual specification of objectives wandering through a given personal space.

The second approach to intelligent design systems also should feature mechanisms which support the *process* of design itself. These mechanisms usually record the intermediate stages of design by generating *scripts* that describe sequences of design procedures. Scripts can be helpful because a variety of design steps can automatically be recorded under a single label, which can then be recalled when a similar design process is required again (Luger and Stubblefield, 1989).

2.2 Sound synthesis as a design problem

We define the process of music composition as *organising sounds that form structures that make sense*. Furthermore, composition is not only considered to be the combination of pre-existing sounds, it also involves an effort to elaborate the sound material (that is, creating the sounds themselves, rather than merely composing with existing sounds (Risset, 1992; Di Scipio, 1994)). "Organisation that makes sense" implies criteria, principles or rules upon which organisational decisions are made (Roads, 1980).

Nowadays, there are no invariable boundaries between sound synthesis and the composition of music. A composer might either think of an evolving single sound that is in itself a piece of music, or of a combination of several discrete sound events. Nevertheless, we prefer not to think of a system for the composition of music here, but to limit ourselves to the design of discrete sound events.

As we said in our introduction, when synthesising sounds to be used in a piece of music, musicians have an intuition about the possibilities of the organisation of these sounds into a musical structure. In attempting to obtain the desired sound, the composer explores a variety of possible solutions, trying out those possibilities within his personal aesthetic (Roozendal, 1993). This process of exploration frequently results in inconsistencies between the composer's best guess at a solution and the formulation of the requirement. If no solution meets his requirement then this requirement either has no solution at all, or it must be redefined. Sound synthesis is seen in this context as a design problem which demands on the one hand clarification of the requirement and on the other hand provision of alternative solutions. Suppose that a composer wants a sound that has high pitch: in order to produce this sound, the system might need the expression "high pitch sound" to be clarified by enunciating that

"high pitch" actually means a fundamental frequency above a certain threshold. If the system still does not understand the clarification, then some sound at least should be produced, which would give some chance that the sound produced may satisfy the user's requirement.

The Department of Artificial Intelligence of Edinburgh University has conducted extensive research in AI-based design systems. Most of our theoretical background in AI-based design is based upon Edinburgh's work; for example, the concept of design as an explicit kind of intelligent behaviour (Smithers et al., 1989) and the use of Newell's three levels of system's study: knowledge, symbolic and engineering levels (Newell, 1981).

The EDS (Edinburgh Design System) was under development in Edinburgh University and several successful experimental prototypes have been implemented. EDS is a system to aid the design of mechanical engines. Mechanical design was of particular interest in the AI department, given the emergence of its advanced robotics assembly research. The mechanical design process is heavily based on geometry; the difficulty with this is to devise formal methods to represent and reason about geometric shape, space and relationships between individual components and the complete mechanical engine. Several techniques to approach this problem have been devised and tested; for example, Smithers et al. (1989) reports on the use of a mixed constraint-based representation of the spatial and topological relationships between geometric features (such as surfaces and edges), in order to formulate geometric reasoning procedures.

The main problem of a system for mechanical design is however to devise intelligent control and management for the design of complete mechanical engines. On one hand the system must possess knowledge of material and manufacturing of individual components and, on the other hand, it must be aware of manipulations, structural combinations and the functional role of the components necessary to assemble a complex mechanism.

Sound design is epistemologically different from mechanical design; for example, while mechanical engines need to function and perform their tasks efficiently, sounds do not need to function, but rather fulfil an aesthetic requirement. Moreover, we are considering a system for the design of discrete sound events and not of complete pieces of music. This is not to say that sound design is simpler than mechanical

design. Even considering that our problem domain is perhaps limited to the design of single sounds, we address several different and no less difficult issues; for example, aesthetic judgement, the use of user-customised pseudo-natural language for communication and intuitive descriptions of sounds which may not initially have clearly assigned characteristics.

2.3 Desirable capabilities of an Intelligent System for Sound Design (ISSD)

By an "intelligent system", we mean a system which works in co-operation with the user, providing useful levels of automated reasoning to render the synthesis tasks less laborious and tedious (tasks such as calculating an appropriate stream of synthesis parameters for each single sound - see Appendix III) and to aid the user to explore alternatives when designing a certain sound. The desirable capabilities of an ISSD are discussed below.

2.3.1 Response to intuitive sound descriptions

We mentioned in our introduction that the specification of the synthesis parameter values for the production of a desired sound is tedious and time-consuming, because the composer has to communicate his ideas to the computer with streams of numbers and low-level programming languages. It is therefore desirable to provide a means of intuitive communication with the computer, as opposed to the specification of quantitative numerical values and low-level programming. We expect a system which allows the musician to design sounds in terms of his own vocabulary of sound descriptors (for example, by using words in English). The system should be able to interpret the meaning of a sound description that uses this vocabulary in terms of the synthesis parameter values necessary to produce the sound.

2.3.2 User configuration

If a system is to allow communication in terms of the user's intuitive sound descriptions, then this system must also have the ability to be configured, or modelled, according to certain basic premises, ranging from the user's familiarity with the sound

world to his own vocabulary of sound description. Thus the instrument (the synthesis algorithm) and the vocabulary of sound descriptors should be user defined. One cannot expect that all composers have entirely the same cognitive processes, nor the same vocabulary to speak about sounds.

2.3.3 Intelligent aid for exploration

The manipulative power of the computer is aimed at supporting the user's exploration of possible alternatives when designing a sound. We believe that numerical settings hardly stimulate the exploration of the possibilities of a synthesis algorithm. Our own experience teaches us that in such a situation we tend to either limit ourselves to a few experiments (it seems that we tend to force ourselves to be pleased with any outcome in order to avoid the specification of new settings), or we try several settings in a random fashion, or both.

In this case the system is expected to have explicitly represented knowledge of qualitative sound attributes and what they mean in terms of sound synthesis parameters. The system is also expected to "know" how one sound may be related to, or distinguished from, another sound. This knowledge is used by the system to create new sounds based upon existing sounds, that is, sounds "known" by the system.

It is desirable to have a system that features mechanisms for knowledge inference and machine learning (see §2.3.4), so that it can help the user to explore this knowledge and encourage the process of collaboration between the user and the computer. Through an intelligent exploratory mechanism, the system should be able to:

(a) suggest possible solutions for a requirement

(b) allow the user to consult the system's knowledge

(c) aid the user in concept formation.

2.3.4 Ability to learn new information from user interaction

Philosophical considerations aside, it is generally recognised that the ability to learn (to adapt, to modify behaviour, to broaden knowledge) is a prerequisite for any form of intelligence (Carbonell, 1990; Suppes and Crangle, 1990).

It is desirable that the system can update its knowledge automatically, in order to cope with new requirements. The system has to start with some basic knowledge of how to synthesise certain sounds and then to learn about other sounds through user interaction.

An ISSD is also expected to be able to assist the user in concept formation, that is, to make generalisations (or classificatory rules) out of its own knowledge base or from external training data given by the user.

2.3.5 The uncertainty factor

We believe that the combination of uncertainty and constraints constitutes a productive ground for generating new creative ideas. Creativity seems to function as an interplay between freedom and discipline. Freedom involves some degree of uncertainty.

The system must be able to invoke some degree of uncertainty in order to create contexts which augment the chances of something unexpected and interesting happening, for example, an unimagined sound from an ill-defined requirement.

2.4 The features our system intends to offer

We pointed out in §2.3 some of the capabilities of an ISSD, which we regard as essential. Considering these capabilities, we indicate the features we intend to offer in our case study system:

(a) the ability to operate the system by means of an intuitive vocabulary instead of sound synthesis numerical values. (It must be said however that the coherence of this

vocabulary depends upon several factors, ranging from how the user sets up the synthesis algorithms, to the terms (or labels) of the vocabulary of sound descriptors.)

(b) the ability to customise the system according to particular needs, ranging from defining which synthesis technique will be used, to defining the vocabulary (for example, English words) for communication

(c) the encouragement of the use of the computer as a collaborator in the process of exploring ideas

(d) the ability to aid the user in concept formation, such as the generalisation of common characteristics among sounds and their classification according to prominent attributes

(e) the ability to create contexts which exhibit some degree of uncertainty.

2.5 The evaluation of the work

We distinguish two classes of approach to computer systems evaluation: namely *subjective evaluation* and *objective evaluation*. A subjective evaluation can assess the external effect of the system's behaviour (it addresses questions such as: "What is the impact of the system on users?", "Does it do what it is expected to do?"), whereas an objective evaluation can assess the inner functioning of the system in terms of its engineering efficiency (it addresses questions concerned with the implementation of the system, such as: "How does it do what it does?", "Is there a more efficient way of doing it?").

In essence there are no standard methods to evaluate computer systems. Even the term "evaluate" has different connotations among researchers in fields such as Computer Science, Cognitive Science and AI (Laurent, 1992). We have, however, defined our own criteria of evaluation.

Since software engineering is not the main concern of this work, we focused upon the subjective evaluation approach. Our main concern is to address the questions "Does the system fulfil the desirable capabilities of an ISSD?" and "What can we learn from this research and from its underlying concepts?". In order to answer the first question,

we invited a number of potential users to play with the system and asked them to report their experience. We made an effort to draw conclusions in as systematic a way as possible, by studying their reports. To answer the second question, we considered the desirable capabilities introduced above and attempted to evaluate our work by indicating its strengths and weaknesses. These will be dealt with in Chapter 8.

2.6 Summary

We began this chapter by introducing a knowledge-based approach for the development of an ISSD. This approach studies the problem on three levels: the knowledge level, the symbolic level, and the engineering level.

We then focused upon the idea of sound synthesis as design. We indicated that the problem of synthesising sounds for a piece of music can be studied as a design problem. We also discussed how we think the computer should be used, in order to aid this process.

Towards the end of this chapter, we presented the features we intend to offer in our case study system and discussed how we evaluate the work.

In the following chapters we examine the implementation of a case study ISSD. We believe that through the design of a particular case study we will be able to effectively examine the problem, in order to facilitate the general development of ISSDs. If we succeed in the construction of a prototype featuring the aforementioned characteristics, then we will have undoubtedly progressed towards a new and useful paradigm for sound design software.

Chapter 3

3 Related work

In this chapter, we present four relevant systems and current research work which have inspired our project. The particular aspects which have promoted this inspiration are assessed in terms of the desired characteristics of an ISSD, as described in the previous chapter.

3.1 The Upic system: a graphic-based system for sound design

The term "graphic-based system for sound design", signifies a system in which the user designs sounds by drawing visual figures.

There are currently various graphic-based systems for sound design, ranging from graphic interfaces for a specific synthesis technique (Orton, 1988), to systems which map geometric curves to various synthesis algorithms (Brandão and Nascimento, 1991). One of the most popular graphic-based systems in Europe at the moment is the Upic system, devised by the composer Iannis Xenakis, in France (Xenakis, 1992).

The idea of the Upic system originated in the Fifties when Xenakis began to write orchestral music using graphic notation to represent sounds (usually large *glissandi*) that were difficult to represent with traditional staff notation (Marino, 1990; Marino et al., 1993). The process of the transcription of graphic notation (comparable to architectonic drawings) into traditional musical notation, so that the music could be performed, is time-consuming and difficult. So Xenakis devised a computer system to interpret the graphics.

The Upic system offers the musician a notation based upon a set of graphic objects. Each object has a specific sound synthesis function designated by the composer, within a fixed framework. The composer has reasonable control on the synthesis process, ranging from the micro compositional level (for example, the form of the wave and the development of the amplitude of the sound) to the macro compositional

level (for example, the duration of the sound event represented by a kind of "vectorial" figure).

When designing sounds using the Upic system, the composer draws a collection of *pages* (see (Lohner, 1986) for a user's report). A page is a set of pitch-versus-time figures, such that pitch corresponds to the vertical axis (from bottom to top) and time to the horizontal axis (from left to right). The composer then "orchestrates" the page by assigning different sets of parameter values to each figure. For example, each figure of a page can be considered as a sound event. Each of the sound events is assigned to a wave table (that is, one period of the sound) and a dynamic envelope. A collection of pages constitutes a score, each page of which is stored separately from the neighbouring pages and may be manipulated at will.

Upic is not, however, a machine that interprets anecdotal drawings: the drawing of a bird, for example, would not necessarily produce a whistle-like sound. The aesthetics of the drawings do not therefore guarantee similar sound quality. The composer must obey certain Upic heuristics and constraints in order to achieve the desired sound. Furthermore, it is not always obvious what these heuristics and constraints are; for example, the user can draw two waveforms which look very different but which sound very similar.

Unfortunately, the Upic system does not interpret the input in real time. First we draw the line, then we hear the sound. This slows down the design process, as pitch, for instance, is particularly hard to co-ordinate without immediate feedback. If, for example, one considers a determined pitch, then it is easy to conjecture where the line should be drawn, but it is unlikely that one would draw it correctly until trying it a few times.

With regard to AI-based systems for sound design, the Upic system does not fulfil all the desirable capabilities of an ISSD (as discussed in Chapter 2). The system has no *knowledge* of waveforms. It is hard to draw everything by hand; even with practice it is difficult to draw waves of a predictable outcome. Alternatively, one can use sampled waveforms from recordings. Even then, however, it is difficult to alter the waveforms or to produce others modelled upon them.

Upic does not take into account the need to represent and integrate the knowledge used in sound design (Smithers et al., 1989). If there is no knowledge explicitly

represented, then there will not be computer collaboration to explore design alternatives. Moreover, with no knowledge integration, there is no means for automatic concept formation, which is essential for the ability to help the user to explore design alternatives. To summarise, the Upic system does not constitute a truly productive ground for exploring and generating new ideas.

We present below an evaluation of the Upic system according to the desirable capabilities of an ISSD, discussed in Chapter 2:

(a) response to intuitive descriptions: does not apply. We wish to refer to sounds by means of a vocabulary of sound descriptors and not by means of visual drawings

(b) user configuration: fair. The user can set-up the wave tables but there are no means for defining the kind of drawings that the user wishes to use for making sounds

(c) intelligent exploratory aid: does not apply

(d) ability to learn: does not apply

(e) uncertainty factor: does not apply. The fact that it is hard to predict the outcome of a drawing is not actually a feature to be proud of. The user must be able to chose when he wants an unpredictable outcome.

We can positively learn from the Upic system that user configuration and response to higher level input (in this case, drawings) constitute two important features of a system for sound design.

3.2 A neural networks-based sound retrieval system

Bernhard Feiten and Stefan Günzel (1993; 1994; Feiten and Ungvary, 1990) propose a neural network approach for the organisation and retrieval of sounds in a data base. They have developed a technique for extracting prominent features of sounds which are then used to represent these sounds on a two-dimensional map.

The system is based on a neural network called Kohonen Feature Map (KFM). The KFM neural network allows the mapping of an *input space* onto a self-organising topology-preserving feature map (Kohonen, 1980). In this case, the input space is a set of sounds and the result is a two-dimensional Sound Feature Map (SFM) which classifies the sounds according to certain detected features. The SFM can then either be used as a sound archive retrieval index or to control a synthesiser which resynthesises the original sounds.

The technique involves a Discrete Fourier Transform (DFT) analysis of the sounds (Dodge and Jerse, 1985) whose data is then mapped onto the vectorial representation required by the KFM algorithm (Feiten and Günzel, 1993). The KFM algorithm classifies the sounds according to a distance measure formula applied to these vectors (each vector representing a sound). Similar sounds are plotted onto neighbouring areas in the map whereas different sounds are separated by greater distances. The map legends are specified by the user, that is, the user gives labels to SFM structures. These labels are then used to retrieve the sounds.

The system presents serious problems when working with dynamic sounds as opposed to simple, steady-state sounds. The problem is that we cause a significant increase of the vector's dimensions by expanding the sound space to dynamic sound - this increases the level of difficulty for classification. Another problem involves the length of the sounds. The dimension of the input vectors for a KFM must be constant, necessitating that the sounds length must also be constant. Insert and delete operations are also problematic here: they are difficult to handle due to the nature of the training process, as the re-organisation of the data base and SFM take too long.

To a certain extent, the system provides the means to intuitively refer to sounds. However, the use of intuition is restricted to giving a label (provided that this label is an "intuitive" one) to the SFM structure of a certain sound given to the system. It does not allow the retrieval of a sound by describing its attributes, rather, it works by using the label given to the SFM structure as a retrieval index.

The system does, however, have a good mechanism for identifying sound features and for classifying the sound according to these features. Although this technique was primarily developed for optimising sound retrieval from a data base, this feature could be adapted for use in a system to retrieve a whole class of sounds. This would give the user more than one choice when requiring a sound with a certain characteristic.

We present below an evaluation of Feiten and Günzel's neural networks-based system according to our required capabilities discussed in Chapter 2:

(a) response to intuitive descriptions: poor. One cannot retrieve sounds by describing their features

(b) user configuration: poor. The user can only give labels to sounds

(c) intelligent exploratory aid: does not apply. This could be improved if provided with higher level symbolic processing

(d) ability to learn: scores very well, but in a different perspective. That is, it does not provide us with the means to access what has been learned (for example, there are no rules to consult)

(e) uncertainty factor: does not apply, but this paradigm could be useful for retrieving the classes of sounds of which the outcome is less predictable.

Similar work is under development by Pierro Cosi and co-workers (1994). They have attempted to develop a timbre classification system based upon auditory modelling and self-organising neural-networks. Although such a system is not primarily aimed at sound design, it demonstrates that substantial data reduction is possible for sound representation. Research into auditory modelling has great potential for raising a new paradigm for sound representation. This would allow us to create a more perceptually-oriented tool for sound analysis. It would therefore facilitate further work into defining descriptors for a sound, by examining the output of its analysis, rather than by referring to the parameters of a certain synthesis algorithm. In this case, the idea of attaching a label to a SFM structure to be used as a retrieval index could also be expanded to sound attributes.

We can positively learn from this neural networks-based system that a pure *connectionist approach* (Todd and Loy, 1991) is not suitable for devising an ISSD. It might however be valuable, if provided with higher level symbolic processing techniques. Antonio Camurri and co-workers (1992) for example, propose a hybrid system that combines symbolic and sub-symbolic processing in an expert system for musical composition, called Harp. The AI area of hybrid systems is currently undergoing much new research (for example, (Kasabov and Petkov, 1992)).

3.3 The Elthar program: a natural language-based system for signal processing

By a natural language-based system, we mean a system in which the user communicates with the computer by means of sentences and expressions in English.

There have been some attempts to create musical sound signal processing systems that understand natural language. The most successful systems function as interfaces for programs which perform tasks related to studio techniques of audio recording: for example, Cims (Schmidt, 1987) and Elthar (Garton, 1989).

Elthar is a sound signal processing expert system designed to interpret natural-language requests from the user. The system has a library of digital signal processing algorithms which are used to process a sound file. In Elthar, the signal processing algorithms mimic recording studio tasks such as mixing, equalising, and multitracking. The user should work with Elthar as if he was working in a recording studio. In addition to the natural language interface, Elthar also features the ability to learn how to use these signal processing algorithms by "observing" how they have been used previously. Once the system has learned how to perform a certain task, the user can re-activate the same task, even if it is an incomplete request, if he wishes. If, for example, a request lacks the parameter values for a certain algorithm, it may be understood by Elthar if these parameters were already given in a previous usage of the same algorithm.

The natural language interface of Elthar scans sentences (or fragments of sentences) and then tries to compute which actions are to be performed. Once the actions have been identified, it activates the respective signal processing algorithms (stored in a library) and processes the sound file. A sentence basically contains an action name (an index to retrieve the signal processing algorithm), a reference to the sound to be processed, a name for the newly generated sound and the algorithm's parameters. The system features a data base of synonyms for action names and parameters so that references to a particular request are very flexible. Instead of numerical values, the user can specify signal processing parameters with adjectives. The adjective *loud*, for example, might be used as a synonym for an *86 dB* value.

Elthar "learns" how to use signal processing algorithms by retaining given parameter values each time an action is performed. These are then used to update a data base of parameter values. The system constructs and maintains a probability distribution which is automatically used to designate values for adjectives when necessary. Elthar also automatically stores *scripts*, that is, it keeps groups of sentences input by the user under a single command name, which can then be recalled in different situations.

The Elthar program features many of the desirable capabilities of an ISSD. It provides efficient mechanisms for knowledge inference and encourages collaboration between the user and the computer. It also has the ability to acquire knowledge from user interaction, but it does not actually assist the user in concept formation. Elthar also provides a good basis for the creation of contexts that augment the element of surprise: for example, once the system has learned how to apply an algorithm on a certain kind of sound, the same algorithm may be re-applied in the same manner, but to a completely different sound. In such a case, the outcome could be fairly unpredictable.

The system is not flexible enough to be configured according to the user's model of sound production, although he can create his own customised dictionary of synonyms. Furthermore, we consider that sound design must primarily be grounded in sound synthesis rather than sound transformation. Our aim is at synthesising a sound from its attributional description. Naturally, transformation is also needed, but we consider that it does not constitute the only point of departure. We present below an evaluation of the Elthar program according to the desirable capabilities discussed in Chapter 2:

(a) response to intuitive descriptions: poor. There is no description for the sounds themselves

(b) user configuration: satisfactory. The user can at least create his own dictionary of adjectives for referring to the algorithms

(c) intelligent exploratory aid: scores well. The user can try various algorithms until the desired effect is achieved. The user can also activate these algorithms by means of incomplete requests (that is, parameters can be missing in a request)

(d) ability to learn: scores well (only if we consider the automatic generation of scripts as a machine learning task; Chapter 4, §4.5.3). It does not however learn about sound synthesis. Moreover, it does not assist the user in concept formation.

(e) uncertainty factor: scores very well. A script applied to a certain sound might produce an uncertain outcome if applied to another kind of sound.

The Elthar program teaches us three positive matters: that is feasible to devise a sound design system that understands natural language-like statements, that *machine learning* is essential to improve the performance of the system and that the uncertainty factor adds to the appeal of the system and motivates user exploration. On the other hand, we can learn from the negative responses that user configuration should be allowed in an ISSD. This feature is important as it enables the user to customise the system according to his specific needs.

3.4 SeaWave: a timbre description-based software for sound design

By a timbre description-based software, we mean a program in which the user designs sounds with adjectives which describe timbre.

SeaWave is an additive synthesiser in which sounds can be designed with a (circumscribed) vocabulary of descriptive terms (Ethington and Punch, 1994).

Sounds are created using the additive synthesis technique, which functions by simultaneously producing a set of sinewaves, called partials. Each partial has its own synthesis parameters, such as frequency, phase, starting point, duration and amplitude envelope. SeaWave provides two levels of communication for the specification of these parameters: a low-level *editing window* and a high-level *transformation window*.

The user has direct access to the synthesis parameters of each partial of the sound at the low-level editing window. In this case, the user designs a sound in terms of the number of partials of that sound, the numerical synthesis parameters for each partial and the amplitude envelope of each partial.

At the high-level transformation window, however, the user does not have access to those low-level parameters: instead, SeaWave provides a menu of adjectives to be selected by the user. The system then maps these adjectives to the respective synthesis parameter values in order to produce the sound they should describe.

A sound in SeaWave is described by three perceptual categories, related to its evolution in time:

(a) *attack*, which describes the onset of the sound

(b) *presence*, which describes the sound while it is sustained

(c) *cutoff*, which describes how the sound ends.

As a result of a series of experiments with various subjects, Russ Ethington and Bill Punch worked out a list of seventeen adjectives and distributed them into those three categories. Attack, for example, can "value" *blown*, *bowed*, *hammed*, *keyed*, *plucked*, or *struck*. They also determined a vocabulary which specifies the degree to which each of the three perpetual categories should be applied: for example the term "slightly more" can be combined with "bowed" to mean *slightly more bowed attack.*

Significantly, at the high-level transformation window, the user does not design a sound from scratch; a "reference sound" (created beforehand at the low-level editing window) is needed. Therefore, the vocabulary actually describes the transformations to be applied to a sound and not the sounds themselves.

For each transformation, the user must select six words from the menu: degree of attack, attack, degree of presence, presence, degree of cutoff, cutoff. For example, a certain sound could be designed by making another sound *slightly more bowed, less bright, and much more damped.*

Although of limited scope, the system offers an excellent interface to edit sounds using adjectives. Ethington and Punch carefully worked out a short vocabulary which can describe reasonably well certain perceptual sound transformations.

As far as the desirable capabilities of an ISSD are concerned, SeaWave proffers good insights but it also presents serious deficiencies.

The system has knowledge of additive synthesis, but does not allow it to be user-modelled or expanded. That is to say, it works only with the limited vocabulary provided by its programmers. It therefore scores badly in the *user configuration* and in *the ability to learn* criteria. Moreover, the user must always specify six terms for the

design of a sound. Another drawback of SeaWave is the fact that a sound is described by only three perceptual categories.

To summarise, we present below an evaluation of the SeaWave system according to the desirable capabilities discussed in Chapter 2:

(a) response to intuitive description: scores reasonably well. Its vocabulary, however, describes sound transformation and not the sounds themselves

(b) user configuration: poor. It works only with additive synthesis and with a fixed vocabulary provided beforehand by its programmers. The only thing the user can customise is the "reference sound"

(c) intelligent exploratory aid: does not apply. Since the knowledge of the system is not dynamic and of limited scope, apparently there is no demand for intelligent exploration

(d) ability to learn: does not apply. There is no possibility for expanding the knowledge of the system

(e) uncertainty factor: fair. Although of predictable outcome, certain sound transformation might cause some interesting surprises.

We can positively learn from SeaWave that it is both possible to define a timbre space for setting the parameters for sound qualities and that it is also feasible to specify a vocabulary to describe these qualities. Moreover, it demonstrates that it is possible to relate this vocabulary to the parameters of a synthesis algorithm.

3.5 Summary

In this chapter we have presented and assessed four pieces of work related to our own research. In each of them we have identified positive and negative aspects related to the desirable capabilities of an ISSD.

The Upic system does have some good features. It offers a limited degree of layered data organisation, but it does not, however, completely match our approach to

computer-aided sound design. Upic is not "intelligent" (in the sense discussed in Chapter 2) mainly because it lacks support for knowledge integration and cannot therefore really assist the user.

The only feature that the neural network-based sound retrieval system adds to our discussion is the fact that it offers a very interesting technique to automatically identify certain sound characteristics. Characterised sounds can then be labelled with words, which subsequently are used to retrieve the sounds from an archive. This does not, however, suffice for sound design, as we need ways to address those sound features and not only the sound name.

No doubt the Elthar program is the one that most contributes to and inspires our research. Although designed for a different application from ours, it features many of the characteristics aimed for in an ISSD, namely natural language-like means of communication, knowledge integration (representation and inference), and the ability to acquire knowledge from user interaction. The major drawback of Elthar is that it is a system which only supports sound transformation of existing sounds, rather than sound synthesis. Moreover, it does not allow for user customisation.

The SeaWave system presents an excellent insight into a timbre description-based system for sound design, although it has limited scope. SeaWave works only with additive synthesis and the user can address it on a higher level only by means of a circumscribed vocabulary. SeaWave lacks flexibility: it does not feature, for example, any means to expand its knowledge. In summary, the major drawbacks of SeaWave are that it works only with additive synthesis and that the vocabulary for sound description is fixed and limited.

We conclude this chapter by indicating the features missing in order to fulfil the desirable capabilities of an ISSD. Firstly, apart from SeaWave, they do not allow the design of sounds in terms of attributional descriptions. Secondly, they do not fully allow user configuration, that is, the user cannot model the system according to his own criteria. Finally, they do not have the ability to automatically form concepts about sounds.

In Chapter 4 we will introduce some important background concepts, in order to aid the reader's understanding of the ideas behind our case study system.

Chapter 4

4 Background concepts

In this chapter we introduce some background concepts necessary to understand our approach to sound design and the ideas behind our case study system.

We begin by presenting the notion of a generic musical instrument, then we go on to introduce the reader to the model of sound production selected for implementing the case study ISSD: the human voice. We continue by presenting three cognitive speculations which inspired us at various stages of this work. Based upon these speculations, we present a discussion on the description of sounds from their attributes. Finally, we present some background AI-related concepts we have employed in our research.

4.1 Defining the boundaries of abstraction of a non-existent instrument

In traditional Western music, musicians work with discrete musical elements, such as notes and their duration. Composers are then encouraged to think both of the production of sounds as the multidimensional control of these elements and also the notation of the music in a score by means of symbols. The performer then interprets the score by relating these symbols to gestures on a musical instrument. These symbols also encourage the representation of the structure of musical processes (for example, Xenakis's symbolic composition, in his book *Formalised Music*, Chapter VI (1963; 1971; 1992)).

Western music traditionally has a certain boundary of abstraction which characterises its representation (for example, the abstraction of a sound event as a note). The inner acoustic features of a sound (for example, amplitude of partials and harmonic content) are not directly relevant for traditional performers and composers, who tend to learn only how to obtain the desired results by acting on the control mechanism of the instrument: they learn what actions to perform from symbols arranged in a score, in order to play the music. (Actions and symbols can be regarded here as components of the body of musical knowledge.) Musicians, in this case, are purely concerned with

the arrangement of symbols in a score, which may require further interpretation since some sound qualities (for example, timbre) cannot be explicitly represented within the limitations of traditional notation.

Contemporary techniques of orchestration do encourage the creation of unusual timbral effects. There are, however, no obvious symbols to notate the harmonic content of individual instruments. Even if those symbols were to be created, the performer would not know how to interpret them because an orchestral instrument does not have "keys" for changing its harmonic spectrum. In traditional Western music, composers and performers have to work with a conceptual model of individual instruments, which has a certain boundary of abstraction that gives very little room for significant manipulation of timbre. The human voice, however, may be considered an exception to this case. Contemporary composers have achieved a great variety of timbres using the human voice (for example, Luciano Berio's *Sequenza III* (1966)).

These problems teach us that in order to think of an ISSD, one must define suitable boundaries of abstraction upon which the body of knowledge will be based. In traditional Western music, a performer transforms musical symbols into instrumental gestures; similarly, a computer works with sound synthesis parameters, in order to produce a sound (for example, the amplitude and frequency values for an oscillator). Thus, the definition of the device which produces the sound is important.

Pierre Schaeffer, in his *Book I* of *Traité des objets musicaux* (Schaeffer, 1966), proposes a generic notion for the definition of an instrument, which is worth considering when defining the boundaries of abstraction. Schaeffer proposes that any device is a musical instrument if it allows one to obtain a varied collection of sonic objects while maintaining a certain identity. The identity of an instrument is maintained by the permanence of certain concrete sound features, which allow for the variation of other features.

In his interpretation of Schaeffer's theory, Michel Chion (1983) labelled the varying features as *values* and the concrete features which maintain identity, as *characters.* The functioning law of musical sound structures therefore can be defined as maintaining permanence of characters and varying the values.

The problem with a computer is that its possibilities are limitless. It is an "instrument" whose timbre does not exist until a programmer defines its boundaries and constraints.

We propose that the definition of the boundaries of abstraction and constraints must involve the specification of a synthesis criterion that maintains certain common features (*characters*) but allows variations to emerge (*values*).

In summary, the definition of the sound production model gives us a particular boundary of abstraction and constraints. This enables us to define a body of knowledge (for example, symbols and actions) for sound production. In Chapter 5 we will consider how to define this.

4.2 An instrument inspired by the human voice

One of the great strengths of computerised sound design is the potential for the composer to create a wide range of *values* and *characters*. It is desirable to work with a model which allows us to deal with a great variety of sounds (flexibility of values) within a well defined criterion (character).

In our research, we decided to base these criteria on a well known model of sound production: the human voice. The mechanism of the human voice itself can produce a wide range of sounds and computers can produce highly sophisticated imitations of vocal sounds.

Another reason for the selection of the human voice model over the use of more abstract mathematical models (for example, Frequency Modulation, (FM)) or scientific metaphors (for example, fractals and cellular automata) is that we found it to be more suitable at this initial stage of our research. We do not yet have either the experience or corroborated methods for relating the terms of a vocabulary for sound description to the parameters of a synthesis algorithm. We find it far more difficult to devise a suitable method if we depart from such formulae or metaphors. The difficulty arises from the lack of any obvious correlations between the parameters of those formulae and the perceptual terms for the description of their acoustics effect (see Chapter 5, §5.1).

Models of the mechanism of the human voice have been extremely helpful in scientific and artistic research in general. Moreover, it is thought that recognition of vowel sounds are essential for our perception of timbre. Many scientific studies have been developed in this field since the last century, from Helmholtz (1885) to the current

research in cognitive science (Howell et al., 1985; McAdams and Deliège, 1985). Furthermore, many techniques for the synthesis of the human voice have been developed and used in speech research and in musical composition (Dodge and Jerse, 1985; Flanagan, 1984; Kaegi and Tempelars, 1978; Klatt, 1980; Rodet et al., 1984; Sundberg, 1991; Cook, 1993, to cite a few).

In order to build a conceptual model of a relatively complex sound production system, such as the human voice, one must look for inspiration from its physical behaviour. At the moment, however, we are not interested in a perfect computer simulation of the human voice. Thus, rather than producing sounds by using more complex techniques (such as those which describe the fundamental aspects of the phenomenon by using of a set of equations (Woodhouse, 1992; Keefe, 1992)), we have opted to produce them by using a more traditional and less complex *formant synthesis* technique.

Among the various techniques available for the synthesis of formants (Clarke, 1988; Miranda, 1992) we selected the *subtractive synthesis* technique (Klatt, 1980; Dodge and Jerse, 1985; Miranda, 1993). Our rationale for this choice is our belief that it is sensible to work with a technique whose functioning resembles the way we understand the functioning of the human voice mechanism (to be dealt with in: §4.2.1, Chapter 5 and Appendix I).

We would emphasise however, that although our approach is currently oriented towards a synthesis method using subtractive synthesis, we expect to be able to expand the results of this investigation to other synthesis models.

4.2.1 Synthesising human voice-like sounds using subtractive synthesis

Subtractive synthesis creates tones from complex sound sources by sculpting selected portions of their spectrum. With this technique, a source with a broad spectrum (usually a white noise or a narrow pulse) serves as the raw material from which we can obtain the desired sound.

4.2.1.1 The source-filter model

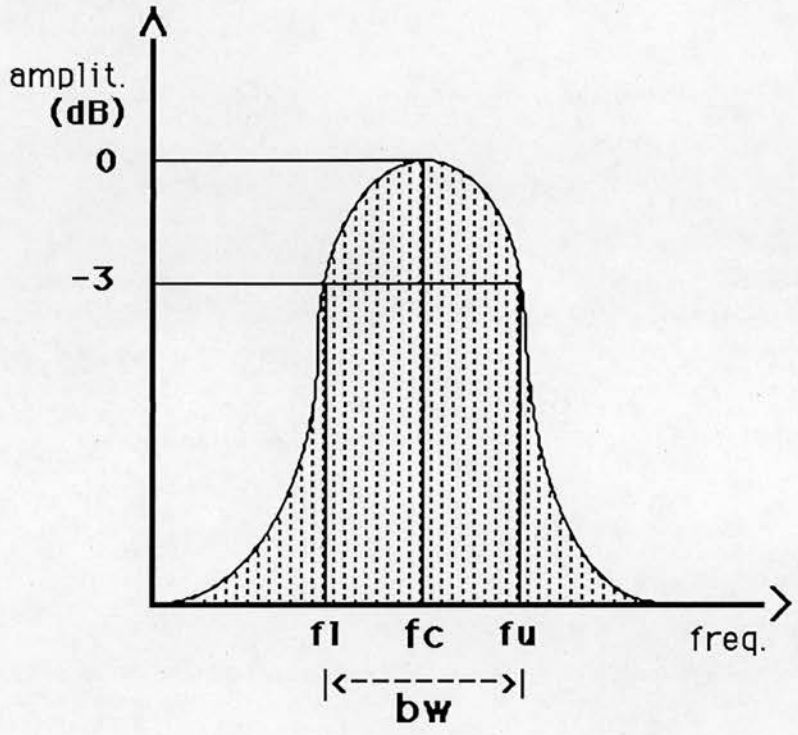
Subtractive synthesis works by using an acoustic model, known as the *source-filter model*, illustrated in Figure 4.1. According to this model, a sound is produced when an acoustic device is excited by mechanical energy, causing the device to transform the excitation. The excitation is called *the source* and the device *the filter*. In the human voice, the source corresponds to the action of the vocal folds, whereas the filter corresponds to the cavity system formed by the larynx, the pharynx, the nasal cavity and the mouth (Johnston, 1989).

Figure 4.1: Source-filter model.

The basic building block of the subtractive synthesiser is the Band-Pass Filter (BPF), also known as the *resonator*. The resonator is a filter which rejects both high and low frequencies with a passband in between. Two parameters are used to specify the characteristics of a BPF: **fc** (passband centre frequency) and **bw** (resonance bandwidth). The **bw** parameter comprises the difference between the upper (**fu**) and lower (**fl**) cut-off frequencies (Figure 4.2).

Under special conditions a BPF may also be used as a Low-Pass Filter (LPF) or as a High-Pass Filter (HPF). A LPF can be simulated by setting the BPF's centre frequency to zero. The resulting cut-off frequency would intuitively be one-half of the **bw**. As a consequence of this simulation, however, the cut-off frequency of the resulting low-pass is 70.7% of one-half of the specified bandwidth; that is, to simulate a LPF with cut-off **f**, we need a BPF with **bw=f*$\sqrt{2}$**. For example, if the desired LPF cut-off frequency is to be 500 Hz, the bandwidth value for the BPF must be 1 kHz multiplied by 1.414 (that is, 1000*1.414). This is because the BPF is a two pole filter; at its cut-off frequency (when output is 50%) the output power of a true LPF of the same cut-off would be in fact 70.7%.

Figure 4.2: BPF is a filter that rejects both low and high frequencies with a passband in between.



4.2.2 Obtaining formants using filter composition

The desired spectrum envelope of a human voice-like sound has the appearance of a pattern of "hills and valleys", technically called *formants* (Figure 4.3). In subtractive synthesis each formant may be associated with the passband centre frequency of a BPF. Thus a composition of filters with different responses is needed in order to obtain such a pattern.

There are two basic combinations for filter composition: *parallel connection* and *cascade connection* (also known as *serial connection*) .

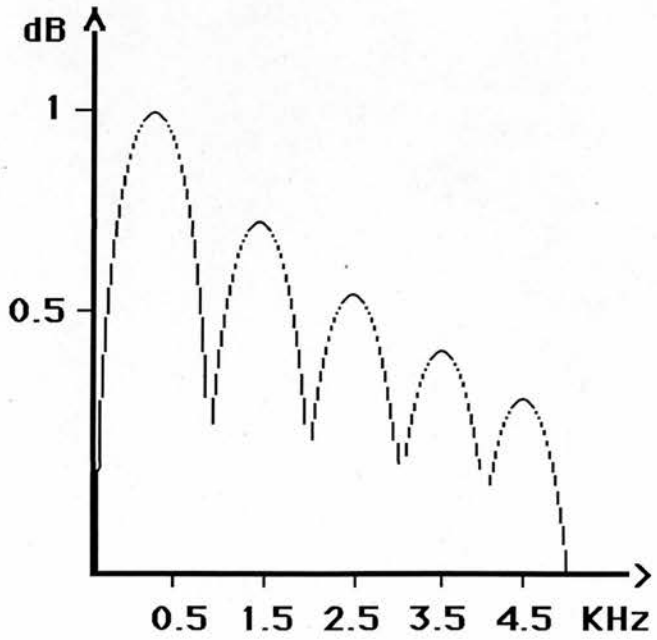
When two or more filters are connected in parallel, the signal to be filtered is simultaneously applied to all filter inputs. Their output is then added together. The parallel connection adds the frequency responses of all filters, resulting in the resonation of any frequency found within the passband of the connected filters (Dodge and Jerse, 1985). Each filter is preceded by an amplitude control, which determines the relative formant's amplitude of the resulting spectrum envelope.

In a cascade connection, filters are connected like the links of a chain. The output of one filter feeds the input of the next, and so on. The output of the last filter is therefore the output of the entire cascade (Dodge and Jerse, 1985). Much care must be taken when composing cascade-connected filters with different passband centre frequencies. Unlike parallel connection, the specification of one filter's passband does not guarantee that there will be significant energy passed in that passband. If any of the previous elements of the cascade has significant attenuation in that frequency range, the response will be prejudiced.

The cascade connection is a more accurate model of the transfer function of the vocal tract, during the production of vowel-like sounds. The parallel connection is however, suitable for generating sounds which violate the usual amplitude relations between formants and for producing unusual speech-like sounds.

Detailed information of the signal processing of our model can be found in Appendix I. In the following paragraphs we introduce some speculative background concepts, followed by a study of the creation of a vocabulary for sound description.

Figure 4.3: The spectrum envelope of a human voice-like sound.



4.3 Three cognitive speculations

By "cognitive speculations" we refer to speculations on how the human mind would organise and use its knowledge of sounds.

We present three cognitive speculations, which have inspired us in many stages of our work. They will help us to define the abstract structures of the system, to define a method for sound description and to devise mechanisms for handling this knowledge.

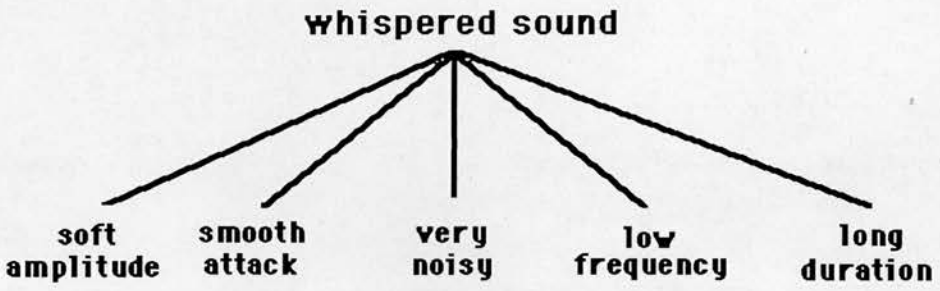
We should emphasise that we do not attempt to develop an accurate model of a composer's cognitive processes for sound design. Rather we use a pragmatic approach intended to identify only those particular cognitive aspects that we think can be simulated using AI techniques.

4.3.1 The layered organisation of knowledge

In this cognitive speculation we borrow some concepts from the field of *knowledge representation* (Bench-Capon, 1990; Luger and Stubblefield, 1989; Brachman and Levesque, 1985). We speculate that humans tend to use a layered approach to the organisation of knowledge - including knowledge of sounds. It is believed that when one thinks of a sound event, he tends to identify its perceptual qualities and regard them as kinds of assembled perceptual units. Together, these perceptual units form a concept in one's mind. This concept is part of his knowledge of the sound world and it is connected to other concepts of the same domain through appropriate relations (McAdams, 1987; Lerdahl, 1987).

In consequence of this speculation, we believe that people tend to represent information of sounds in a layered manner and use the higher level layers to recall them. Taking as an example, a *whispered* sound: one might associate it with attributes like *soft amplitude, smooth attack, very noisy, low fundamental frequency, and long duration.* (The meaning of these terms is clarified in Chapter 5.) In this case, we would say that humans tend to group this information and recall it simply as *whispered* instead of as *soft-smooth-noisy-low-long* (Figure 4.4).

Figure 4.4: Representing knowledge in a layered manner



This speculation is worth consideration, as it is a well-known phenomenon that learning and memory are strongly enhanced if one can organise information in a hierarchical manner (Dowling and Harwood, 1986; Sloboda, 1985).

4.3.2 The generalisation of perceptual attributes

We speculate that, when one listens to several distinct sound events, he tends to characterise them by selecting certain sound attributes which he thinks are important.

When listening to several distinct sound events, it seems that the human mind prioritises the selection of certain attributes which are more important in order to make distinctions among them. We say that in this case humans tend to make a *generalisation* of the perceptual attributes.

If one carefully listens to a sound event, there will probably be a large number of possible intuitive generalisations. It is therefore indispensable to select those generalisations we believe to be appropriate. These depend upon several factors such as context, sound complexity, duration of events, sequence of exposure and repetition, which make a great variety of combinations possible. Humans are, however, able to make generalisations very quickly; perhaps because we never evaluate all the possibilities. One tends to limit one's field of exploration and resort to an heuristic (§4.4.2.1). We believe that this plays an important role in imagination and memory when creating sounds and composing with them.

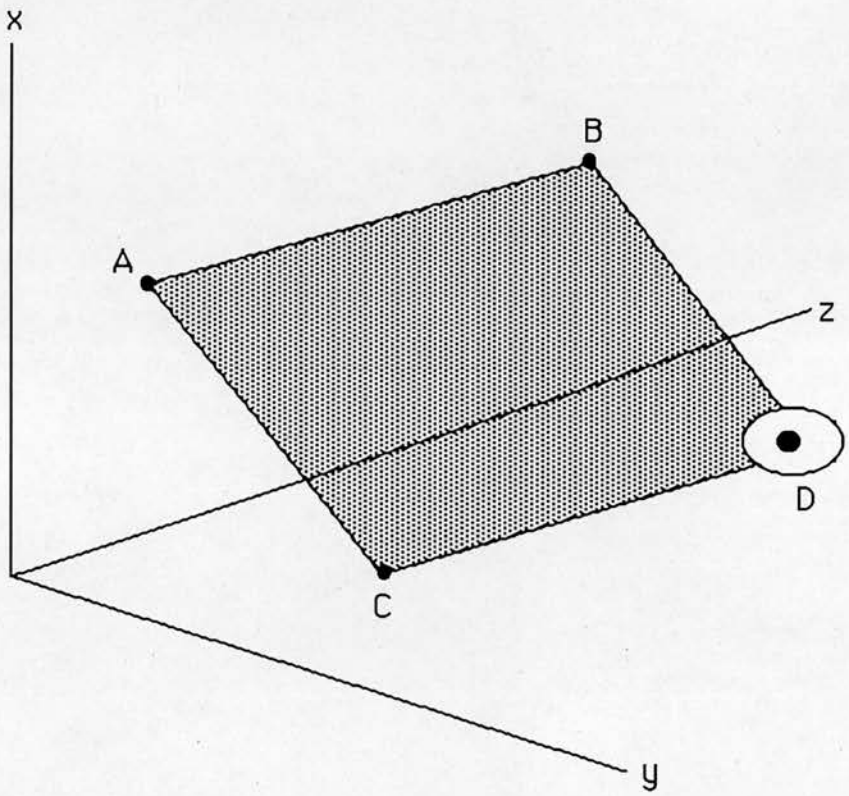
4.3.3 The identification of sound analogies

We also speculate that, in some cases, a timbre analogy can be defined within a n-dimensional *attribute space*. This speculation is influenced by several experiments by David Ehresman (1978) and David Wessel (1979), at Ircam, Paris. The basic idea is illustrated in Figure 4.5. If, for example, we form a three-dimensional space from three attributes (so that each attribute varies continuously); given sounds A, B and C, whose three attributes correspond to co-ordinates of this space, there is a sound D, such that A, B, C and D constitute the vertices of a parallelogram. In this context, sound D is expected to be perceptually related to sounds A, B and C, as follows.

Let us suppose that sounds are described by three attributes corresponding to the axes **y, x,** and **z**: first formant centre frequency (**f1**), second centre formant frequency (**f2**), and fundamental frequency (**f0**), respectively. On the one hand, **sound(a)** and **sound(b)** may have similar values for axes **y** and **x**, that is, **f1 = 290 Hz** and **f2 = 1870 Hz**, whilst on the other hand, **sound(a)** and **sound(c)** have the same value for axis **n**, namely **f0 = 220 Hz**, and the value of **f0** for **sound(b)** is equal **440 Hz**. If one makes a two-sound pattern - **sound(a):sound(b)** - and wishes to make an analogous sequence beginning on **sound(c)**, then according to this speculation the best choice would be the sound whose attributes best complete a parallelogram in the space, that is, **sound(d)**. In perceptual terms, what has varied is the attribute pitch: **sound(b)** sounds higher than **sound(a)** but preserved the same **f1** and **f2**. Therefore in order to obtain an analogous two-sound pattern beginning on **sound(c)** (say **f1 = 650 Hz** and **f2 = 1028 Hz**), the best solution would be **sound(d)**, with **f1** and **f2** inherited from **sound(c)** but **f0** inherited from **sound(b)**.

Although this speculation can work well for certain optimal cases (for example, it might not suit larger dimensions), we find it encouraging that a systematic description of sound events is both possible and meaningful. Furthermore, it provides a model for the generic notion of instrument introduced in §4.1. We could therefore say that **sound(d)** *preserved* the same formants of **sound(c)** but *varied* the value of its fundamental frequency.

Figure 4.5: Identifying sound analogies.



4.4 Describing sounds by means of their attributes

In this section we study how we can define a vocabulary of attributes for sound description. Several studies have identified a framework to systematically describe sounds from their attributes (Bismark, 1971; 1974a; 1974b; Cogan, 1984; Giomi and Ligabue, 1992; Terhardt, 1974; Schaeffer, 1966; Wishart, 1985). They are mainly derived from work in the fields of both psychoacoustics and musical analysis. It is not our aim to survey all these, so we have selected two examples for discussion. The first example illustrates how one can specify some attributes to describe those sounds produced by the source-filter model introduced in §4.2.1. The second example illustrates how one could identify attributes to describe sounds in a more general manner, independently of a specific model of sound production; it also provides an example of a means of organising and representing the knowledge of perceptual attributes.

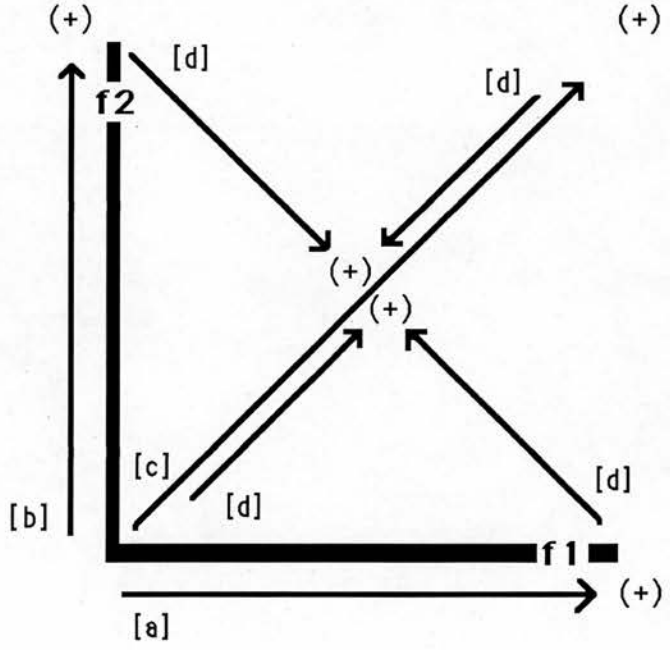
4.4.1 The sound quality of formants

To reiterate, the source-filter model (§4.2.1.1) proposes that the characteristic of a sound is determined by its spectrum envelope's pattern. This pattern is composed of multiple "hills" called formants. Each formant has a centre frequency peak and a bandwidth.

According to Wayne Slawson (1985), the two lowest formants are the most significant determinants of sound quality.

The pattern of the spectrum envelope of formant frequencies is considered to be the result of a complex filter, through which a source sound passes.

Figure 4.6: Two-dimensional sound space.



[a] = openness
[b] = acuteness
[c] = smallness
[d] = laxness

By considering the "identification of sound analogies" speculation, we can define here a two-dimensional space whose axes are the first (**f1**) and the second (**f2**) formant centre frequencies, respectively. Four perceptual attributes, *openness*, *acuteness*, *smallness*, and *laxness* (Slawson, 1985; 1987), can be specified, as categories of equal-value contours in this space. The attribute openness varies with **f1**; acuteness with **f2**; smallness with both **f1** and **f2**; and laxness tends towards a neutral position in the middle of the space (Figure 4.6).

Considering Figure 4.5, §4.3.3, if we say that **f1** corresponds to axis **y** and **f2** to axis **x**, then we can conclude that, in perceptual terms, **sound(b)** seems higher than **sound(a),** but both preserve the same openness and acuteness.

This example illustrates what we understand by the "permanence of characters" and the "variation of values" of an instrument, discussed in §4.1.

4.4.2 The concept of maintenance

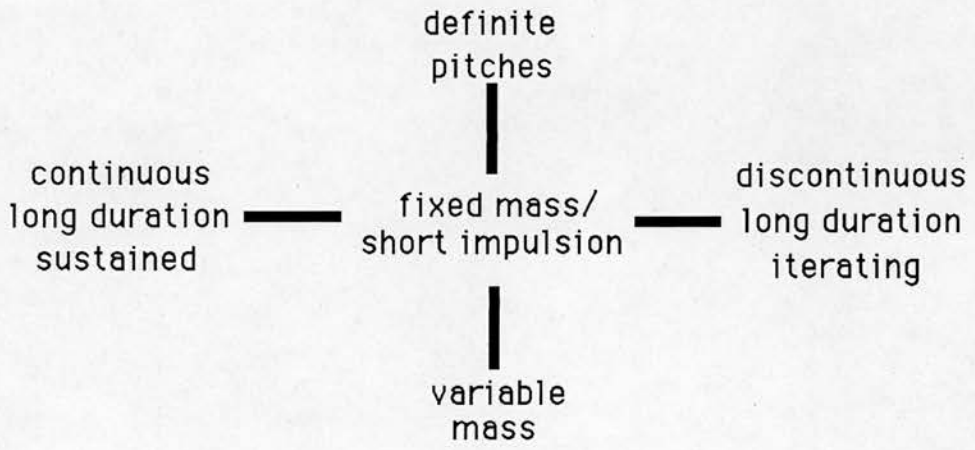
In *Traité des objets musicaux* Pierre Schaeffer (1966) proposes a theory of sonic objects which purports to define criteria for the identification and classification of sounds according to general features. He also describes the effect of these features upon possible emerging musical values. The concepts of typology and morphology are the crux of this work, but their boundaries are unclear on first sight. We would say that typology is concerned with the identification of discrete sound objects from a sound continuum, whereas morphology seeks to actually classify them according to their qualities.

As we cannot summarise the complexity of the whole *Traité* in a few paragraphs (Palombini, 1992; 1993a; 1993b), we have selected only one of its aspects to study here: the concept of *maintenance.* Schaeffer observed that sounds result from a certain energetic process, called *maintenance* (or *continuation*, according to the interpretation of Trevor Wishart (1985, pp. 97)), which describes the development which a sound undergoes with time. If the sound is merely ephemeral, a non-resonant single sound such as a drum-stroke or a vocal plosive consonant, then there is a discrete *short impulsion.* If the sound is prolonged, such as a sung vowel, then there is a *continuous*, that is sustained sound. If it is prolonged by the repetition of impulsion, such as a drum-roll or the rolled repetition of the consonant "r" in Spanish or in

(Brazilian) Portuguese, then there is *iterating* maintenance. As an example, one could obtain an iterating sound, by specifying a very low frequency, such as 8 Hz, to a pulse generator (Wisnick, 1989). In contrast, a continuous sound could be produced by setting a higher frequency, such as 128 Hz, to this pulse generator (Appendix I).

A two-axis diagram (Figure 4.7) can be defined. The middle horizontal axis contains sounds of *short impulsion*, the left features the *sustained sounds* and the right has those whose maintenance is *iterating*. On the vertical axis, sounds with *fixed mass* (or fixed noise-band) are placed between sounds of *definite pitches* and sounds of *variable mass*. Most sounds produced on acoustic musical instruments settle extremely fast on a fixed pitch or on a fixed noise-band (or *pink noise*), following the attack stage. *"Mass"*, in this case, can be considered as the amount of information carried by the sound, ranging from redundancy (a meagre steady sinewave of fixed pitch) to total unpredictability (white noise). These perceptual phenomena may be obtained by inputting noise into a BPF (§4.2.1.1) whose passband is shortened (this tends to produce a sound with definite pitch) and widened (with results in a sound with variable mass).

Figure 4.7: The maintenance diagram.



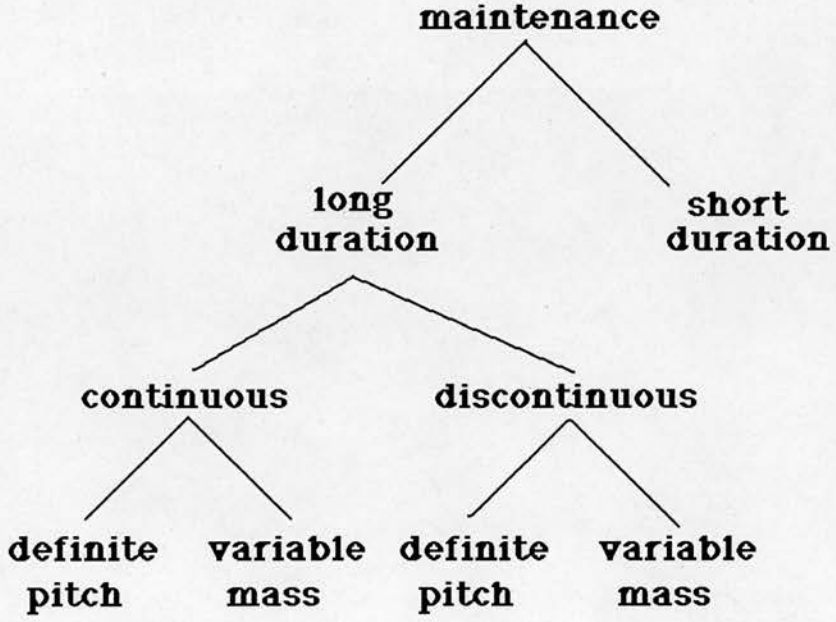
This example has not been implemented in our case study system. It is not yet obvious how one could implement it using the technique we developed (we refer back to this issue in Chapter 8, §8.3).

4.4.2.1 Organising and representing the knowledge of sound maintenance

Bearing in mind the "layered organisation of knowledge" cognitive speculation, we examine how one could organise and represent the aforementioned knowledge of maintenance into a kind of taxonomy (Figure 4.8).

Suppose that one had a sound in mind (for example, the *whispered* sound of §4.3.1) and wished to describe it by means of the concept of maintenance. If duration is considered the most salient perceptual attribute, then if the sound had a long duration, the next generalisation would involve thinking whether the sound would be sustained (that is, continuous) or iterating (that is, discontinuous). Finally, one would consider whether the sound had a definite pitch or variable mass. In this case, the attributes of the *whispered* sound would be: **long duration**, **sustained** and **variable mass**.

Figure 4.8: Organising and representing the knowledge about sound maintenance.



This example illustrates what we consider to be the organisation of knowledge of sounds in a layered manner. It provides an insight into how one might develop a framework to describe sounds.

4.5 A preliminary introduction to background AI concepts

In this section we present the background AI concepts we have employed in our research. We reinforce knowledge representation issues, present the basic theory of *machine learning* and *knowledge acquisition*, and indicate the techniques we have used and their role in our approach to building ISSDs.

4.5.1 Knowledge representation

Designers of knowledge-based systems require knowledge representation schemes that provide representational power and modularity. A knowledge representation scheme must capture the knowledge needed for a problem solution and provide a framework to assist the system's designer to easily organise this knowledge.

We present below a brief discussion on knowledge representation and introduce the technique used in our investigation.

4.5.1.1 The internal representation hypothesis

Our primary assumption is that mental activity is mediated by internal representations. Although there is no consensus as to what these representations actually are (some regard them as neurophysiological states, whilst others may define them as symbols or even images), we have chosen the most traditional AI symbolic approach (Bench-Capon, 1990; Luger and Stubblefield, 1989). This approach assumes that intelligent activity is achieved through:

(a) the use of symbols to represent a problem domain

(b) the use of these symbols to generate potential solutions to problems

(c) the selection of a suitable solution to a problem.

The use of an adequate knowledge representation technique is therefore one of the most important keys for the design of successful AI systems.

4.5.1.2 A brief survey of knowledge representation paradigms

4.5.1.2.1 Logic representation: first-order predicate calculus

A number of logics have been developed in philosophy and mathematics to represent knowledge; for example *propositional calculus* and *first-order predicate calculus*. The first-order predicate calculus is largely used in AI systems.

The first-order predicate calculus provides a well-defined language for describing and reasoning about qualitative aspects of a system. It can denote objects of a domain by using simple symbols and can express relationships between objects, assertions and denials of these relations, and logical relations between these statements (Luger and Stubblefield, 1989).

The first-order predicate calculus is sufficiently general to provide a foundation for other models of knowledge representation. AI problem domains however often require large amounts of highly structured interrelated knowledge. Some high-level notion of structure is needed to help the system's designer represent complex concepts in a coherent way. The first-order predicate calculus alone does not provide this help.

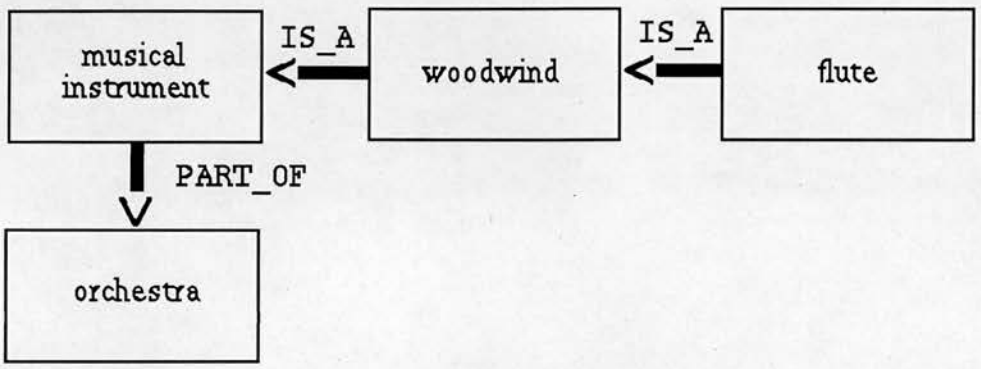
4.5.1.2.2 Network representation: graphs

A network representation also provides the means to denote objects of a domain and relations between them by using simple symbols. The advantage of network representations over logic representations is that the former can provide some high-level notion of structure that helps the system's designer to represent taxonomically structured information. The philosophy behind a network representation is that one reasons about a concept or object by relating it to other concepts or objects of the domain.

Graphs technique are an example of network representation; it provides a means to explicitly represent objects and relations by using *nodes* and *arcs*. A number of graphs techniques have been developed and used in AI systems; for example, *conceptual graphs* and *semantic networks*.

A semantic network, for instance, consists of a network of nodes linked by arcs, so that nodes represent concepts, or objects, and arcs represent relations between them; both nodes and arcs are usually labelled (Figure 4.9).

Figure 4.9: An example of a simple semantic network.



In Figure 4.9, there are 4 objects ("musical instrument", "orchestra", "woodwind" and "flute") and 2 types of relations ("part_of" and "is_a"). This semantic network represents the following facts:

A *musical instrument* is part of the *orchestra*.
A *woodwind* is a *musical instrument*.
A *flute* is a *woodwind*.

Other facts can be inferred from the network, in addition to the facts which are explicitly represented. For example:

A *flute* is part of the *orchestra*.

One of the advantages of a graph-based representation is that facts come from the definition of links and associated inference rules that define specific mechanisms, such as the inheritance mechanism. In the case of the above example, "flute" inherited the fact that it is "part of" "orchestra".

In itself a graph-based notation of relationships is not so different from the first-order predicate calculus. The power of a network representation is that it provides an

explicit method to represent objects and relations, and promotes the organization of knowledge into class hierarchies and the inheritance mechanism.

4.5.1.2.3 Structured representation: frames

Network representations allow for the representation of knowledge using explicit links between single objects in a knowledge base. Structured representations however extend network representations by providing a means to organise large networks of knowledge into a collection of separate networks, each of which representes some stereotyped situation or class of objects.

Frames technique is a type of structured representation. Frames technique allows for the representation of complex structures by encapsulating multiple attributes of situations, or objects, into single units, or classes of objects, in the domain.

A frame is a data structure whose components are called *slots*. Slots have names and accommodate various types of information: a value, a link to other frames or procedures to calculate its value. A slot may also be left incomplete.

As in semantic networks, the most useful feature of frames is the inheritance mechanism; when a frame represents a class of objects and another frame represents a superclass of this class, then the class frame inherits from the superclass the values for its incomplete slots. Examples of frames:

FRAME: ***musical instrument***
part_of: ***orchestra***

FRAME: ***woodwind***
is_a: ***musical instrument***
excitation_method: ***air stream***
resonance_method: ***pipe***

Note that slots are similar to the arcs of a network representation. Slots however have the advantage that they can hold procedures to perform some function, in addition to links to other concepts.

Structured representations thus extend network representations by representing complex objects as interconnected strucured single entities, rather than as one single large network.

4.5.1.3 The schema approach

In our investigation we adopted a representation approach called *schema* (Seifert, 1991). Schema is a type of structured representation. Our rationale to adopt the schema approach is inspired by Stephen McAdams' concept of *schemata* (1987).

Stephen McAdams indicates that the form of a representation is important in that *"different forms of representation, even though they carry the same meaning, may have different properties with respect to what the process of transformation can do with them"* (pp. 18), and that, *"certain aspects of reality are more apparent in some sorts of representation than in others"* (pp. 18). McAdams also introduces the concept of *schemata*, possibly inspired by Kantian philosophy (Oliveira, 1981). McAdams regards schemata as structures which combine symbols that create concepts and establish the relations between them. Memory may therefore be represented as a kind of storage, that is itself a model of the evolving physical world. The knowledge that has been acquired from the world is represented by the state of the model at any given moment. This model of the perceptual process is viewed as the construction of a representational schema. It is assumed that *" ... precepts are pre-fabricated building blocks that are derived from experience. ... a schema is a pattern for assembling perceptual units or other schemata into larger structures or unitary wholes. And, finally, these schemata can operate on various levels to discern structures in the sensory information either at a level of expressive variation or at a level of global form."* (McAdams, 1987, pp. 23).

By studying McAdams' concept, we conclude that an attributional description of a sound is based upon a kind of "sonic image" of the sound's contours in a "phenomenal field", which helps us to identify such attributes. We found the schema approach to be the most suitable to represent a "sonic image". The notion of schema is studied in more detail in Chapter 6, §6.2.

4.5.2 Machine learning

Machine learning (ML) is a major sub-field of AI, with its own various branches (such as the logic-based approach (Plotkin, 1970), the psychological approach (Laird et al., 1987), the neurophysiology-inspired approach (McClelland and Rumelhart, 1986), and the epistemological approach (Hayes-Roth, 1983)). Perhaps the most popular current debate in ML, and in AI in general, is between the *sub-symbolic* and the *symbolic* approaches (Fodor and Pylyshyn, 1988). The former, also known as connectionism or neural networks, is inspired by neurophysiology; it intends to provide alternative ML mechanisms (that is, with no human intervention) so that the desired computation may be achieved simply by repeatedly exposing the system to examples of the desired behaviour. As the result of learning, the system records the "behaviour" in a network of single processors (metaphorically called "neurones"). The neural networks technique has great potential; at this stage of our research, however, it is not a practical procedure. It is not yet obvious how the computer can provide the support we aim for in sound design, using the knowledge recorded in a neural network. (This will be clarified in Chapter 5. Also refer to the commentary made in Chapter 3, §3.2. For a survey on neural networks-based music systems, see (Todd and Loy, 1992)).

We have chosen the more traditional ML symbolic approach (Winston, 1984). Explicit representation of knowledge is fundamental here, as the user must interact with the system using this knowledge. In essence, the system should start with significant user-defined information in the body of knowledge (which will be expanded, adapted and modified through user interaction), avoiding the need to learn everything from scratch. As Pasteur once suggested: *"In the field of observation, chance favours only the prepared mind"* (quoted in (Tang, 1984, pp. 264)). This can also apply to our approach to sound design.

Several algorithms for symbolic learning are being employed in AI systems. These range from *learning by being told* to *learning by discovery* (Bratko, 1990; Carbonell, 1990). In the former case, the learner is told explicitly what is to be learned by a "teacher". No "teacher" is involved in the latter case. From learning by discovery, the system automatically discovers new concepts, merely from observations, or by planning and performing experiments in the domain. Many other techniques lie between these two extremes; unfortunately there is no ideal machine learning technique

to deal with all dimensions of a domain. The criteria for selecting a machine learning technique depends upon its application. The purpose of machine learning in our system is to induce general concept descriptions of sounds, from a set of examples. The ML technique selected for our investigation is therefore the *inductive learning* technique (Chapter 6, §6.5). The benefit of being able to induce general concept descriptions of sounds is that the machine can automatically use induced concept descriptions to identify unknown sounds, or even to suggest missing attributes of an incomplete sound description.

The aim of inducing rules about sounds is to allow the user to explore further alternatives when designing particular sounds. The user could ask the system, for example, to "play something that sounds similar to a vowel" or even "play a kind of dull, low pitched sound". In these cases the system would consult induced rules to infer which attributes would be necessary to synthesise a vowel-like sound, or a sound with dull colour attribute (Smaill et al., 1994).

4.5.3 Knowledge acquisition

Knowledge acquisition refers to the ability of the system to automatically insert new information about sounds and sound descriptors in its knowledge base.

The distinction we make between machine learning and knowledge acquisition is that the former makes rules from a collection of data, whereas the latter simply adds new data to the system's body of knowledge.

4.6 Summary

In this chapter we have introduced some important background concepts. We introduced the generic notion of a musical instrument. Then, we presented the fundamentals of the instrument which will be used in the implementation of our case study ISSD: a formant synthesiser which uses subtractive synthesis in order to produce human voice-like sounds.

We presented three cognitive speculations which inspired our study of how to devise an ISSD featuring all the desirable capabilities discussed in Chapter 2. The "layered

organisation of knowledge" speculation played an important role in the design of an abstract scheme for representing sounds. The "generalisation of perceptual attributes" speculation motivated the inclusion of machine learning in the system. Finally, the "identification of sound analogies" speculation inspired us to develop a suitable aid for exploration (Chapters 5 and 6).

We continued by presenting a discussion of the description of sounds by means of their attributes. We demonstrated how the cognitive speculations may aid both the identification of perceptual sound attributes and the organisation of this knowledge. We then introduced background AI concepts of knowledge representation, machine learning and knowledge acquisition.

We have learned that in order to design an ISSD, one must first ascertain a means to devise a model of computer sound production which would give us a coherent basis for the specification of a meaningful sound description vocabulary. We demonstrated, for example, that the source-filter model (§4.2.1.1) can provide a good basis for the specification of sound attributes, which may vary according to the values of the first and second formant centre frequencies (§4.4.1). The attribute *openness*, for example, is proportional to the value of the first formant centre frequency: as we increase the value of this frequency, we also increase the *openness* of the sound. In the process of learning how the computer can help us to manage this information, we found that the AI symbolic techniques of knowledge representation, machine learning and knowledge acquisition are suitable.

In the following chapters we expand and employ these concepts for the design of our case study ISSD.

Chapter 5

5 The knowledge level

In this chapter we study the knowledge level of our case study ISSD. At the knowledge level, the system has a simple structure. Its components are: *body of knowledge*, *goals* and *actions*. The knowledge level aggregates what the system must know (for example, information and inference mechanisms) in order to function. Goals are states of affairs to be achieved and actions use the information of the body of knowledge in order to compute these goals. Actions, at the knowledge level, establish interrelations within the system's information. It also connects the memory (which contains the information of the body of knowledge and actions) and the goals of the system, but with no specification of any mechanism through which these relations or connections are established. Actions can also automatically add new information to the body of knowledge.

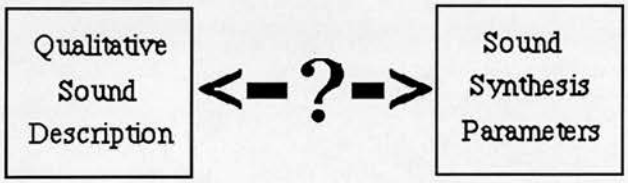
There is some controversy as to the benefit of regarding the knowledge level as a separate computer system level (Winkhuyzen, 1992). Philosophical considerations aside, however, we propose that this level is most useful if we consider it to be a means to introduce the problem under discussion and to anticipate the kind of behaviour we desire in our system. We believe that the knowledge level facilitates the definition and interpretation of the abstract structures embedded in the system (Rademakers and Pfeifer, 1992). We propose that by distinguishing a knowledge level, we would have better means to define the abstract structures of the symbolic level more generally, so that the system (which embodies these structures) may be made to support wider variations, or instantiations, of the system's body of knowledge (Chapter 8, §8.1).

We identify the goal of the system here and the difficulties involved in pursuing it. We go on to identify the body of knowledge needed to achieve this goal. We then propose a model for action (involving knowledge inference and machine learning) and study its behaviour through some hypothetical examples.

5.1 Identifying the goal: the mapping problem

In general, the main problem with the knowledge level of the system in question is the mapping between a composer's intuitive description of sounds and the parametric control of computer sound synthesis (Figure 5.1).

Figure 5.1: Is it possible to map qualitative sound descriptions to sound synthesis parameters?



We have learned from the early stages of computer music that it is not sufficient to solely employ abstract mathematical models as formulae to generate sounds for a piece of music, without accounting for the fact that our ear is culturally determined (for the early stage we refer to works such as Stephen Holtzman's non-standard synthesis (1978), SSP and ASP (Berg, 1975; 1980), and Iannis Xenakis's stochastic sounds, (1963; 1971; 1992)) . Quoting Jean-Baptiste Barrière (1989, pp. 118), *"any approach to sound synthesis must be built as an extension of our collective memory, in a dialectical movement between memory and creativity, tradition and invention".*

Different approaches have been proposed to explain aspects of sound perception, mental representation of sounds and their description. Examples of these approaches range from innate (biological) predisposition (Spender, 1980) and natural morphology (Petitot, 1989; Wishart, 1985) to anthropological archetypes (Bayle, 1993) and *'zoomusicologie'* (Mâche, 1991). It is apparent that most of these aspects depend upon several cultural factors such as personal taste, training, expertise and language. Trained listeners, for example, have a greater awareness of the mental structures they are using. This implies that they have a more extensive vocabulary for sound description which enhances memory capacity. Therefore, it is desirable to design a system with a non-specific vocabulary, which can then be customised by the user.

We must also consider that there are constraints imposed by sound synthesis techniques (formant synthesis, frequency modulation, amplitude modulation, etc.)

which limit the scope of the sound world. One cannot, for example, ask a musician to produce a long, smoothed sound with long vibrato on a harpsichord. Therefore we should not limit the system to only one model of sound production.

Although we have restricted ourselves to a particular sound domain (the domain of human voice-like sounds produced by means of subtractive synthesis) our ultimate aim in this thesis is a system which enables the user to specify his own sound domain. We must enable the musician to explore his own subjectivity within the scope of his own sound world. The system must enable the user to specify his own model of sound production and vocabulary for sound description.

The goal of our system is to produce a sound from an attributional description. The main problem is the mapping of this description to synthesis parameters. The system must know how to accomplish this mapping, or model for action. In order to accomplish this, the system must have appropriate information stored in its knowledge base: the sound synthesis parameters and the vocabulary for sound description. This information should be supplied by the user.

5.2 Specifying the information of the body of knowledge

The information of the body of knowledge of our example study system comprises:

(a) the sound synthesis parameters (and how they work)

(b) the vocabulary for sound description.

We introduce a method to specify the information of the body of knowledge. This method's operation is based upon the idea that the way in which the instrument is designed (that is, the signal processing of the sound production model) dictates the definition of the vocabulary.

5.2.1 Specifying the synthesis parameters

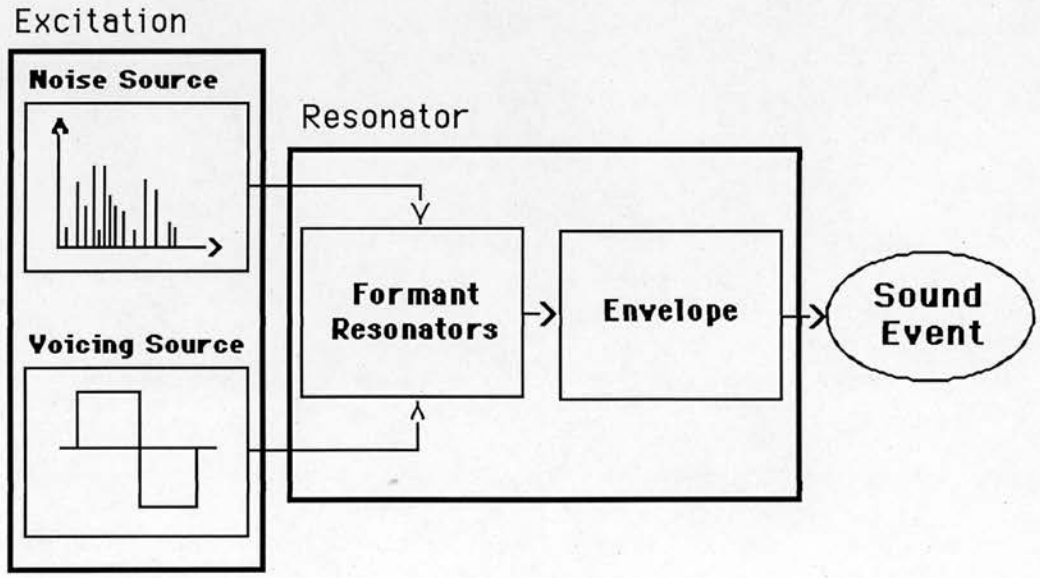
When designing the signal processing of an ISSD, it is important to consider that the definition of its modules (for example, the blocks of the diagram shown in Figure 5.2)

and their connection play an important role for the specification of the knowledge of the system. We believe that it is important to work with a model of sound production, whose functioning may be represented by modules which perform specific tasks and which, in some way, resemble our understanding of the functioning of the model. In this case, note that the source-filter model (introduced in Chapter 4, §4.2) supports modular representation and indeed resembles the way we understand the functioning of the human voice.

We have already discussed in Chapter 4 our selection of the subtractive synthesis technique. Here, we demonstrate why this technique suits our purposes. We then illustrate how we designed the signal processing of our case study ISSD and how we identified the synthesis parameters and studied their functioning. The signal processing we propose for our case study ISSD is illustrated in Figure 5.2.

One of the main reasons for our selection of the subtractive synthesis technique, is that it allows us to design a signal processing architecture whose modules perform specific tasks that can be associated with parts of the human voice mechanism. For example, the sound sources (or excitation) are associated with the vocal folds, the filters (of the formant resonators) are associated with the "tube" that begins at the vocal folds and ends at the lips and so on (Appendix I).

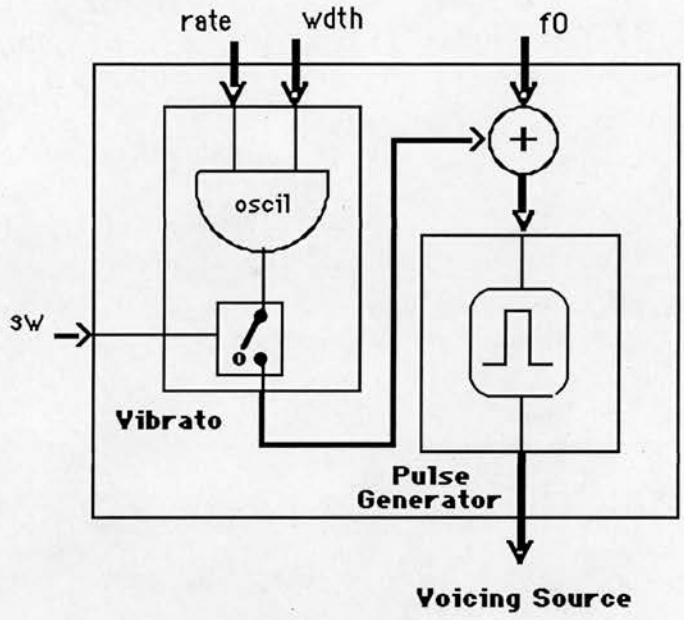
Figure 5.2: The signal processing block diagram.



The design of the signal processing architecture facilitates the creation of the vocabulary for sound description. One could easily define a vocabulary to describe the sounds produced by the instrument shown in Figure 5.2, by referring to the fundamental frequencies emitted by the voicing source, to the spectrum response of the filters and so on. In §5.2.2 we discuss the specification of the vocabulary.

Figure 5.3 exemplifies the specification of the synthesis parameters of the voicing source of the instrument shown in Figure 5.2.

Figure 5.3: The voicing source.



The voicing source module has a pulse generator and a source of vibrato. Each of them needs parametrical values to function:

(a) **f0** = fundamental frequency (of the pulse)
(b) **rate** = vibrato rate
(c) **wdth** = width of the vibrato
(d) **sw** = a switch (to connect the vibrato source to the pulse generator).

We obtain various sorts of voicing source depending upon the values we set for these parameters: for example, by setting **rate** to a frequency equal **5.2 Hz** and **wdth** to a value corresponding to **3.5 %** of **f0**, we obtain a voicing source similar to the one produced by the human vocal folds in singing. The modularity of this signal

processing model facilitates the specification of groups of parameters, whose variation of values carries significant perceptual sound features.

The complete list of the synthesis parameters of our case study ISSD is given in Appendix II. A detailed study of its signal processing can be found in Appendix I. An example of the role of its synthesis parameters can be found in Appendix III.

5.2.2 Specifying the vocabulary for sound description

In Chapter 4, §4.4, we introduced a discussion on the specification of a vocabulary for sound description. To reiterate, by "vocabulary for sound description", we mean a set of user-defined labels that are used to refer to a combination, or group, of synthesis parameter values (termed here as "attributes" and "attributes values"). We illustrate below how to specify this through the study of some examples.

Two approaches can be identified to accomplish this task: a *signal processing influenced approach* and a *signal processing independent approach.* By adopting the signal processing influenced approach, one specifies a vocabulary of sound descriptors by using the signal processing model at hand as a point of departure, for example, using the source-filter model introduced in Chapter 4, §4.4.1. On the other hand, by adopting the signal processing independent approach, one specifies a vocabulary of sound descriptors which are more or less independent of any model of sound production, for example, using the concept of maintenance discussed in Chapter 4, §4.4.2.

To illustrate the signal processing influenced approach, let us tie an attribute labelled **voicing source** to the *voicing source* module (Figure 5.3). We can now define descriptive values (for example, adjectives in English) for this attribute. These descriptive values are to describe the output caused by the values of the input synthesis parameters. For example the "attribute=value" expression: **voicing source = steady low**, if **f0 = 80 Hz, rate = 5.2 Hz, wdth = 3.5 %** of **f0**, and **sw = on**.

We find vectors useful when ascribing attribute values. Taking as another example, the attribute **openness** Chapter 4, §4.4.1. This is an attribute which we have tied to the *formant resonators* module of Figure 5.2. In Chapter 4, §4.4.1, we proposed that

"openness" is a perceptual sound attribute that corresponds to the vector given by the co-ordinate **f1** (Chapter 4, Figure 4.6). The higher the value of **f1**, the higher the openness of the sound. After identifying a vector, the next step is to divide it into discrete sub-vectors such that each of them corresponds to different perceptual stages. For the attribute **openness**, one could define three perceptual stages, each corresponding to a certain first formant centre frequency range of values, for example:

openness = low if **f1 =< 380** Hz,
openness = medium if **f1 > 380** Hz and **f1 < 600 Hz**, and
openness = high if **f1 >= 600** Hz.

It is important to note that we distinguish between two kinds of terms in our vocabulary for sound description: *attribute* and *attribute values*. Attribute is the label we tie to a module of the instrument (or to a "meta-parameter") whereas attribute values are the labels which we give to the various sorts of output of this module, according to the variations in its synthesis parameter values.

In the next section we study how the system can map attributional descriptions of sounds to the respective synthesis parameters.

5.3 Devising a model for action

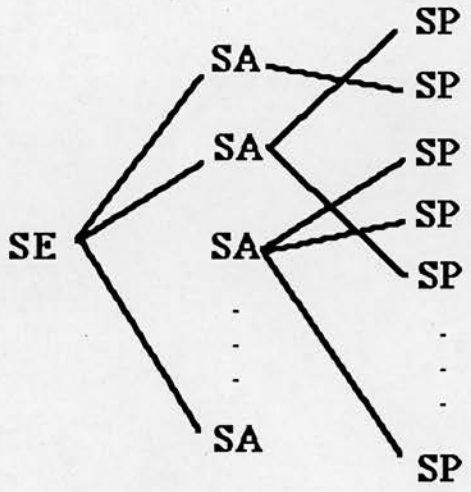
In Appendix III we introduce an example of a vowel synthesis using our proposed model. By examining this example one can see the complexity of the model and the great amount of knowledge and patience needed to set all the synthesis parameters values each time one wishes to produce a sound.

In this section we devise a model for action for our system. The model for action establishes the correlation between the vocabulary for sound description and the synthesis parameters of the instrument: that is, given a sound description, the system has to compute the parameter values needed to produce the sound.

5.3.1 Correlating the vocabulary for sound description and the synthesis parameters

We introduce here the model for action and examine how this model establishes the correlation between the vocabulary for sound description and the synthesis parameters. Our proposed model for action is represented in Figure 5.4. A sound event **SE** is composed of several sound attributes **SA**. Each sound attribute in turn corresponds to one or a set of synthesis parameters **SP**. An action engine is responsible for computing the necessary synthesis parameter values to produce a sound event. The input for the action engine can be either the name of the desired sound event, or a set of sound attribute values. Alternatively, one can input a set of synthesis parameter values or a set containing both sound attributes and synthesis parameter values. The output is a set of synthesis parameter values which in turn are used for synthesising the required sound event.

Figure 5.4: The proposed model for action.



The action engine must be able to compute all the necessary synthesis parameter values for synthesising a sound event. If the input requirement is ill-defined (for example, an incomplete list of sound attribute values), then the action engine must be able to deduce the missing information by making analogies with other sound events which have similar constituents. To do this, the action engine must be automatically able to induce rules of "prominent" sound features, that will identify which sound attributes are more important for describing sound events (see §5.3.2). Furthermore, given that synthesis

parameter values may also be given as part of the requirement, the action engine must be able to handle values which may not match with any "known" sound attribute. Likewise, the action engine must be able to automatically add this new information to the system's body of knowledge.

Let us consider the example given in Chapter 4, §4.3.3. Suppose that the system has the following information in the body of knowledge:

SE = { sound(a), sound(b) }

SA = { openness = { low, high },
acuteness = { low, high },
fundamental frequency = { low, medium, high } }

SP = { f1 = { 290 Hz, 400 Hz, 650 Hz }
f2 = {1028 Hz, 1700 Hz, 1870 Hz },
f0 = { 220 Hz, 440 Hz , 880 Hz} }

The symbols in parenthesis denote possible values for the element on the left side of the equation. For example, a sound attribute **SA** may value **openness**, **acuteness**, or **fundamental frequency**. The sound attribute **openness** may in turn value either **low** or **high**.

The action engine must be able to infer that, for example, **sound(a)** is described by having **low openness**, **high acuteness** and **low fundamental frequency**. Furthermore, it must infer that **low openness** and **high acuteness** mean **f1 = 290 Hz** and **f2 = 1870 Hz**, respectively and that **low fundamental frequency** means **f0 = 220 Hz**.

Inputting only the information **medium fundamental frequency**, is an example of an "ill-defined requirement" to synthesise a sound (for example, **sound(a)**). In this case, the action engine must evaluate the missing information needed to produce **sound(a)**. This can be accomplished by either consulting the induced rules of prominent sound features (the ones that identify which sound attributes are more relevant for describing a certain class of sound events - see §5.3.2), or by inventing a new sound, whose information is then added to the body of knowledge.

Example:

Hypothetical Requirement:

"Play a sound with low fundamental frequency."

The action engine:

The system has already induced some rules (see §5.3.2, below) which suggest that **sound(a)** best matches this description. Thus the system produces **sound(a)**.

In the following paragraphs we suggest how the system can induce rules about sound events and how it can acquire new information about sound events and sound attributes.

5.3.2 The induction of "prominent" sound features and the acquisition of new attribute values

It is desirable that the system should have the ability to induce rules about sounds. A preliminary introduction to inductive machine learning and knowledge acquisition was presented in Chapter 4, §4.5.

Referring back to Figure 5.2 and to the example illustrated above in §5.3.1: by inducing rules of a sound, we mean inducing the shortest set of sound attributes which can be used to either describe a sound event (which the system knows), or to distinguish it from other sound events.

In the above example (§5.3.1), the system has induced that the attribute value **fundamental frequency = low** on its own is enough to characterise the sound **sound(a)** (see comment towards the end of the conclusion of this chapter, in §5.4).

By "acquisition of new attribute values" we mean the ability to automatically add new sound attribute values to the knowledge base, by deducing that a set of synthesis parameter values (input in a request) does not match any sound attribute value known by the system. To illustrate this capability, suppose that the system knows that:

low openness corresponds to **f1 = 290 Hz** and
high openness corresponds to **f1 = 650 Hz.**

If as a requirement, the user inputs a synthesis parameter value equal to **f1 = 400 Hz**, then the system should deduce that it knows no sound attribute value tied to it. In this case the system should do the following:

(a) produce the sound

(b) inform the user that it has deduced a new attribute value

(c) ask the user to label this new information

(d) add the new attribute value to the body of knowledge.

Suppose that we wish to label it **medium**. Then the following information is added to the body of knowledge:

medium openness corresponds to **f1 = 400 Hz.**

Example:

Hypothetical Requirement:

"Play a sound with high acuteness and first formant centre frequency equal to 400 Hz."

The action engine:

The system discovers that there is no attribute value for **openness** that matches with **f1 = 400 Hz.** Therefore the system knows no sound that matches this description and has no information about **openness** if **f1 = 400**. In this case, considering that the user decided to label it **medium**, the system

produces a new sound and adds the new information about it to the body of knowledge, such that:

SE = { sound(a), sound(b), sound(novel) },
...
SA = { openness = { low, medium, high },
...
... }

5.4. Summary

In this chapter we described the knowledge level of our example study system. The components of the knowledge level are: goals, *information* of the body of knowledge and actions.

The goal of our system is to synthesise a sound from its qualitative description. The main knowledge needed is concerned with the mapping between qualitative sound descriptions and the respective synthesis parameters.

The information of the body of knowledge comprises the sound synthesis parameters and the vocabulary for sound description. One of the desired capabilities of our system is the ability to be customised to the user's own understanding of sound synthesis and to be adaptable to his personal vocabulary for sound description. On the basis of this consideration, we concluded that the user should supply the synthesis model and his own sound description vocabulary. This process need not, however, be exhaustive, as the system should have the ability to update its knowledge of attribute and synthesis values through user interaction. Nevertheless, the system should provide the algorithms that compute this information, in order to synthesise the sounds. We also indicated some fundamental concepts concerned with signal processing design and their influence in the creation of the vocabulary for sound description.

Finally, we proposed a model for action. This model describes how the information of the knowledge base can be correlated and used in order to accomplish a goal; that is, what the system has to do with information in order to produce a sound. Inspired by

the cognitive speculations introduced in Chapter 4, §4.3, we devised a model for action which features:

(a) the ability to search for synthesis parameter values, given a certain collection of user defined sound attributes

(b) the ability to make analogies in order to deduce missing attributes in an incomplete, or unsatisfactory description

(c) the ability to update the body of knowledge through user interaction

(d) the ability to make generalisations in order to infer which attributes are "more distinctive" in a sound.

The term "more distinctive" does not necessarily refer to what one would perceive to be the most distinctive attribute of a sound. As we mentioned in Chapter 4, §4.3.2, when we listen to several distinct sounds, we tend to select the attributes which are most relevant; this depends upon several factors, which have a great variety of combinatorial possibilities. We are however, able to make generalisations very quickly, apparently because we do not check all the possibilities: rather, we tend to limit our field of exploration and resort to an heuristic. One of the aims of Artificial Intelligence is to devise a variety of algorithms that behave in similar ways to specific types of heuristics used by humans to make generalisations. In Chapter 6 we introduce a brief discussion of generalisation algorithms and present the algorithms used in our investigation.

In Chapter 6 we also study how to represent the information of the body of knowledge and explain how the action engine works. The role of cognitive speculations in the design of this system will be more evident in Chapter 6.

Chapter 6

6 The symbolic level

In Chapter 5 we described the knowledge level of our case study ISSD, here we will consider its symbolic level.

At the symbolic level we have to devise a scheme to represent the information of the body of knowledge and devise the algorithms for information processing. This scheme includes data structures which hold and connect information, whereas the algorithms extract and use the information contained in these structures. The symbolic level thus involves designing an architecture that embodies the knowledge level.

We begin by presenting the architecture we propose for our system. Next, we introduce the notion of *schema*. A schema is an abstract data structure aimed at mediating attributional descriptions of sounds and their respective synthesis parameters. We then study how to represent the body of knowledge. We also introduce the algorithm that processes this information. We continue by presenting the machine learning and knowledge acquisition algorithms of the system. The chapter ends with some final remarks on the symbolic level as a whole.

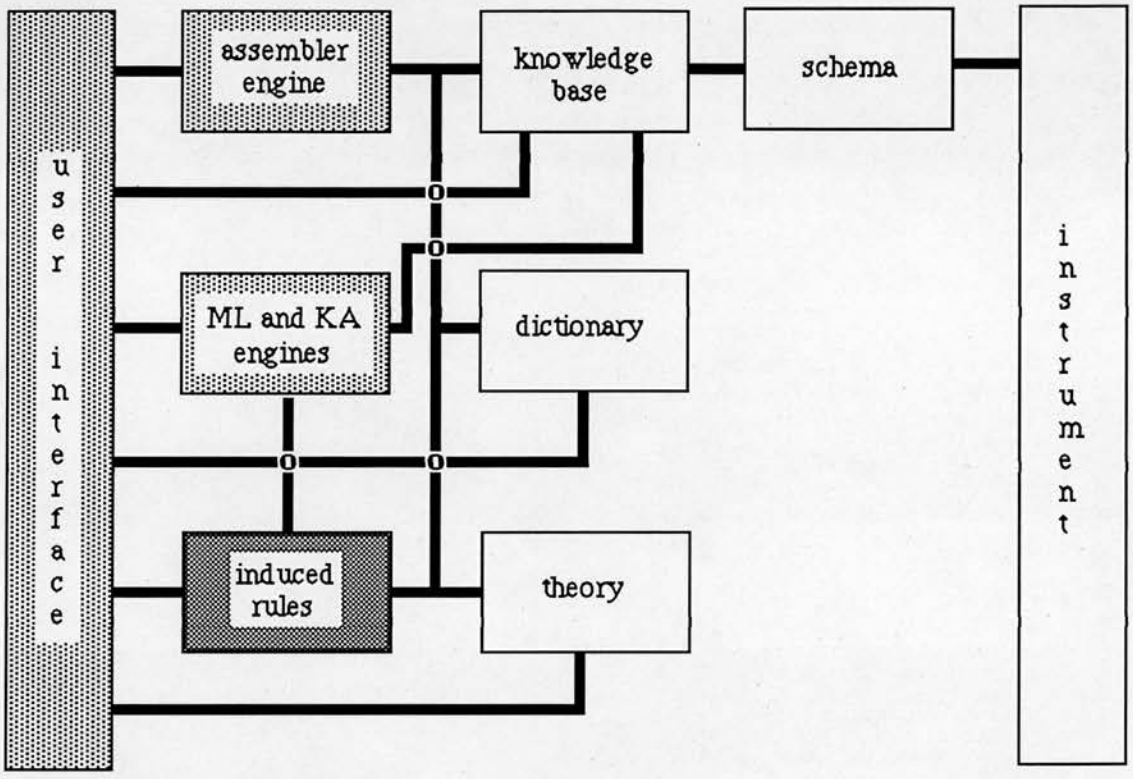
6.1 Towards a system architecture

User configuration is one of the desirable capabilities of an ISSD. Rather than providing a system that reflects a particular sound synthesis model, we propose an architecture featuring open-ended modules (see proposed system architecture in Figure 6.1).

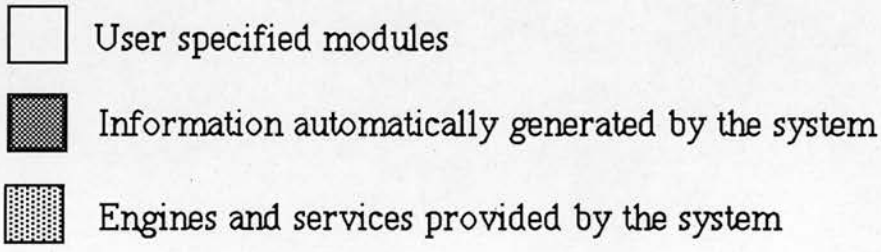
We understand that computer systems are by no means neutral: a piece of software tends to reflect a personal attitude towards the act of creation itself and sound design software is no exception. As Peter Beyls (1993) observed, a computer program is by definition coloured by the subjectivity of its programmer. It is desirable that the programmer and the composer are the same person; however the composer can get

side-tracked in a perpetual programming cycle. As an alternative to achieve a good balance, we propose an architecture which enables the user to customise the system. In general terms, our proposed architecture provides means to handle information whilst remaining open-ended with regard to the nature of the information (Figure 6.1).

Figure 6.1: The proposed system architecture.



The modules of the architecture are classified in three groups:



We present a brief introduction to each module of the architecture; their underlying concepts are subsequently explained. Further details of their implementation and functioning will be discussed at the engineering level, in Chapter 7.

6.1.1 Engines and services provided by the system

The "engine and services provided by the system" modules embody the model for action introduced in Chapter 5, §5.3. The algorithms of the *assembler engine* and of the *ML and KA engines* (for Machine Learning and Knowledge Acquisition) modules will be fully explained later in this Chapter. The *ML and KA engines* module performs two tasks: *inductive learning* and *automatic knowledge acquisition* (briefly introduced in Chapter 4, §4.5 and in Chapter 5, §5.3.2) whilst the *user interface* module provides means to communicate with the system.

6.1.2 Information internally generated and administered by the system

The *inductive rules* module holds the information internally generated by the system as the result of the inductive learning performed by the *ML and KA engines* module.

The user should be able to consult the information contained in this module, at any time.

6.1.3 User-specified modules

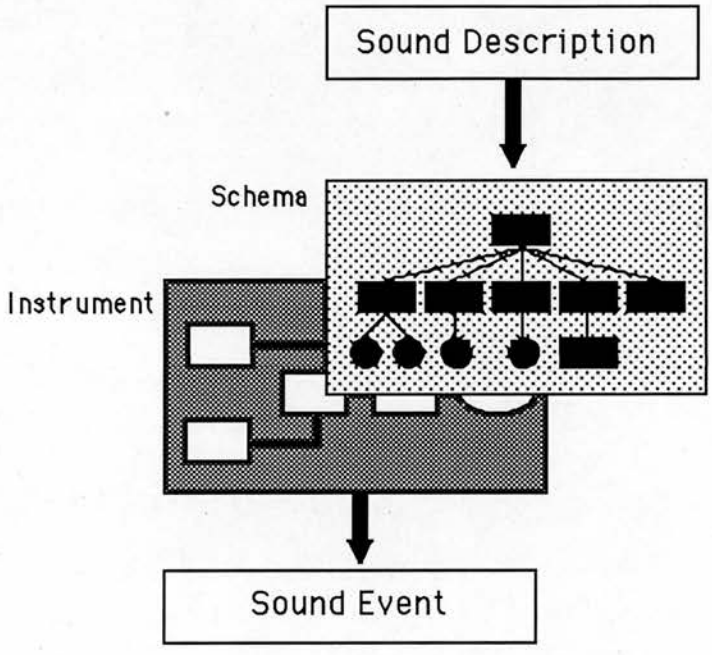
These are the open-ended modules where the information of the body of knowledge is specified (Chapter 5, §5.2). The user-specified modules define the domain of the system, that is, the sonic world of the system. In these modules the user specifies the signal processing of the instrument, the knowledge base whose information is used to "play" it (§6.3), a dictionary of slot values (§6.1.3.3, §6.3.3) and a theory for the instrument (§6.1.3.4, §6.3.3).

6.2 The notion of schema

A schema is a multi-levelled structure aimed at mediating sound descriptions, that is *sound attributes*, created from a user-defined vocabulary of descriptors and their respective synthesis parameters (§6.3.2). The role of the schema is twofold: it should

embody a multi-levelled representation of the signal processing of an instrument and should also provide an abstraction to represent sounds (Figure 6.2).

Figure 6.2: A schema is a multi-levelled structure which mediates sound descriptions and their correspondent synthesis parameters.

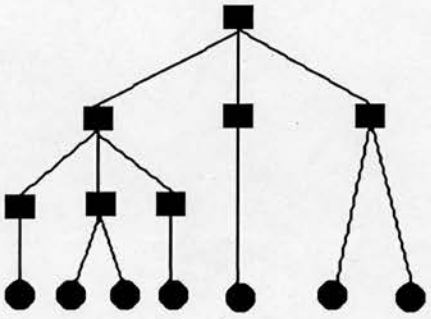


The schema enables the organisation of knowledge of sounds based upon the signal processing model that produces them. A sound event is represented here in terms of the various perceptual features which contributes to its identity. These features must however be tied to the signal processing model in some way (Chapter 4, §4.4; Chapter 5, §5.2.1, §5.2.2). We assume that each descriptive attribute (identified in a sound event produced by an instrument) is caused by a certain component, or a group of components, of this instrument (Chapter 5, Figure 5.2). Let us take as an example an extremely simple instrument, which has only one component: an analogue VCO (Voltage Controlled Oscillator) (Eiche, 1987; Rossing, 1990). If we tie an attribute called **frequency** to the VCO and take the a.c. voltage as its input parameter, the attribute value **high frequency**, could be made to correspond to *a high input voltage value*.

6.2.1 The Abstract Sound Schema (ASS)

The Abstract Sound Schema (ASS) is inspired by the cognitive speculation of "layered organisation of knowledge" (Chapter 4, §4.3.1). It was devised to represent a schema. The ASS consists of: *nodes, slots* and *links*. Nodes and slots are components and links correspond to the relations between them. Both components and relations on the ASS are labelled.

Figure 6.3: The Abstract Sound Schema. Slots are represented as circles and grouping nodes as rectangles. A line represents a link. Slots, nodes, and links are labelled.



Slots are grouped "bottom up" into higher level nodes, which in turn are grouped into higher level nodes and so on until the top node (Figure. 6.3).

The ASS is a tree-like abstract data structure whose ultimate nodes (the leaves) are slots. Each slot has a user-defined label and accommodates either a sound synthesis datum or a pointer towards a procedure to calculate a sound synthesis datum. Higher level nodes also have user-defined labels. They correspond to the modules and sub-modules of the signal processing architecture. The top node (the "root of the tree") corresponds to an abstract sound event.

In order to study how the algorithm for knowledge inference works, we explain how we use the ASS for the representation of knowledge of sounds and attributes. We give a detailed example in Chapter 7 of the multi-levelled representation of the instrument's signal processing architecture.

6.3 Representing knowledge of sounds and attributes using the ASS

In this section we explain how sounds and sound attributes are represented by means of the ASS. We also examine here the way in which we can refer to sounds using a user-defined vocabulary of sound attributes.

6.3.1 The notion of sound assemblage

The ASS provides an abstract data structure to represent a sound event. An instantiation of the ASS corresponds to a sound. The ASS's slots must be "filled" with synthesis parameter values in order to instantiate a certain sound. An instantiation of a sound event is an *assemblage*. For each different sound produced by the instrument there is a corresponding assemblage. Considering the case of an instrument which simulates the vocal tract mechanism (Chapter 4, §4.2), an assemblage would correspond to a certain position of the vocal tract which produces a certain sound. We are not concerned here with changes to the vocal tract during the production of a sound; however, an assemblage could also be made to correspond to a sound which changes its "colour" during production, for example, a diphthong.

In practice, the information for instantiating the ASS is recorded in a knowledge base as a collection of slot values. The information for the assemblage of a particular sound event is clustered around a collection of slot values. To produce a sound (Figure 6.4), an *assembler engine* is then responsible for:

(a) collecting the appropriate slot values

(b) assembling the ASS

(c) activating the synthesis algorithm (that is, the instrument; see also Figure 6.2)

The assembler engine algorithm is given in §6.4.

Figure 6.4: The assemblage engine firstly collects the appropriate slot values in order to assemble the desired sound and then activates the synthesis algorithm.

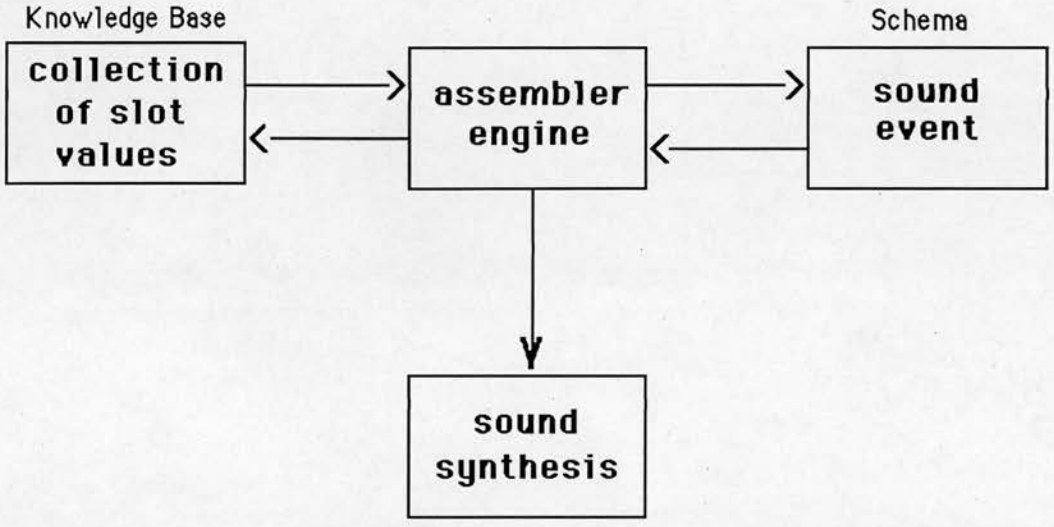
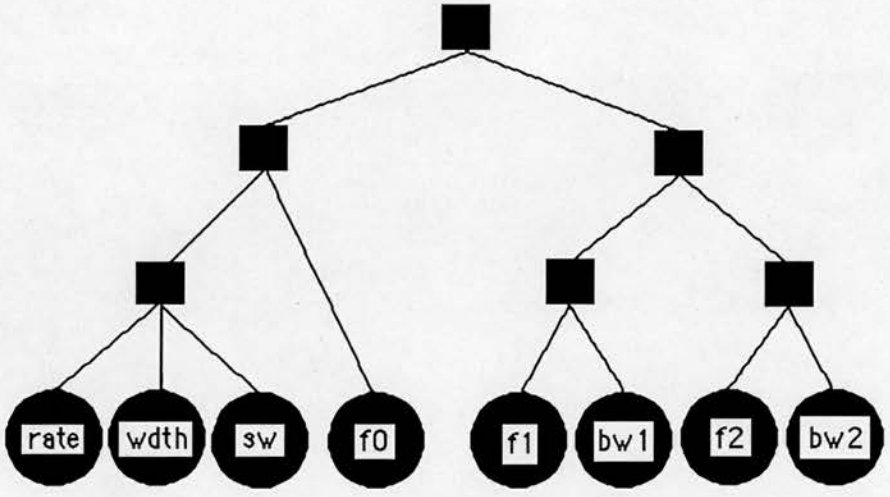


Figure 6.5: A certain ASS.



6.3.1.1 The notion of sound inheritance

Sound descriptions (that is, collections of slots) are recorded hierarchically in the knowledge base. The ability to represent hierarchically the relationship between slot collections is useful for the inheritance relation. Inheritance is a mechanism by which an individual assumes the properties of its class and by which properties of a class are

passed on to its subclass (Luger and Stubblefield, 1989). This hierarchical organisation is accomplished by means of a link referred to as *a kind of* (Figure 6.6). When a slot collection for a sound is related, by means of the link *a kind of*, to another slot collection at a higher level, the former inherits properties of the latter. Note in Figure 6.6 that **sound(b)** is said to be *a kind of* **sound(a)**. This means that slots which eventually may not be defined for **sound(b)** will be instantiated with slot values taken from **sound(a)**. In practice, the assembler engine must "know" that the missing slots in one level are inherited from a higher level (see example in Figure 6.7). Inheritance reduces the size of the knowledge base by promoting the definition of common properties once only; it helps to prevent update inconsistencies and redundancies and it also aids the integration of knowledge.

Figure 6.6: The example knowledge base.

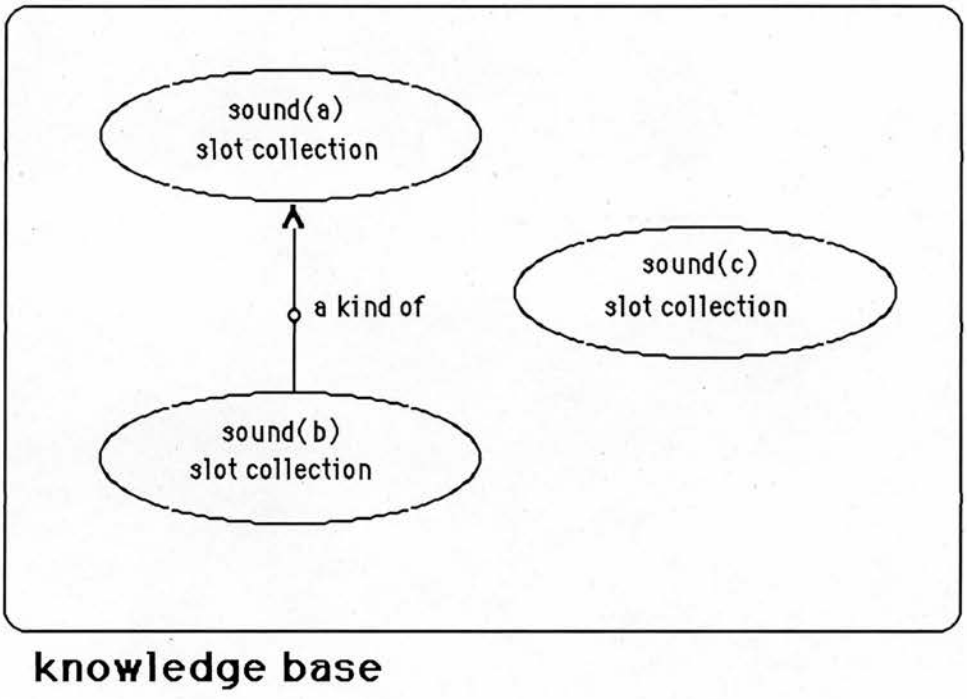
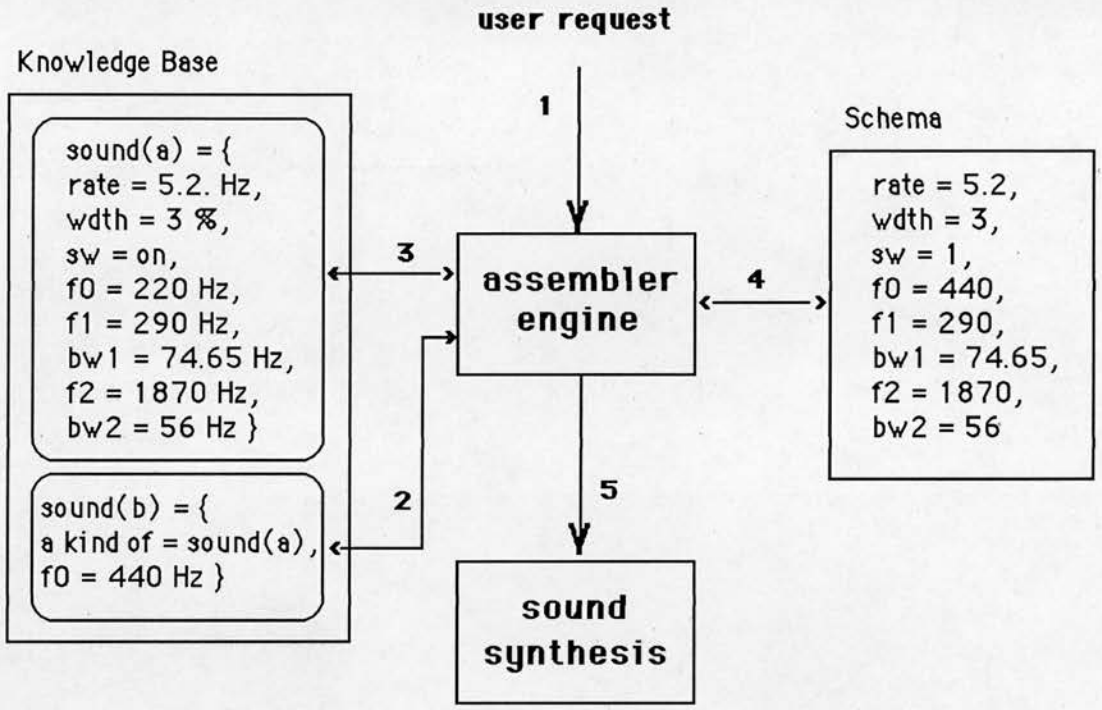


Figure 6.7: An example of the assembler engine functioning which instantiates the schema shown in Figure 6.5 using the knowledge base shown in Figure 6.6.

(**1**) The user inputs a request: produce **sound(b)**.
(**2**) The assembler engine finds **sound(b)** in the knowledge base and collects the value of **f0**.
(**3**) The assembler engine collects the other slot values from **sound(a)**, because of the inheritance.
(**4**) The assembler engine *assembles* the ASS.
(**5**) The assembler engine finally sends the slot values (which correspond to the synthesis parameters of the instrument) to the synthesis algorithm.

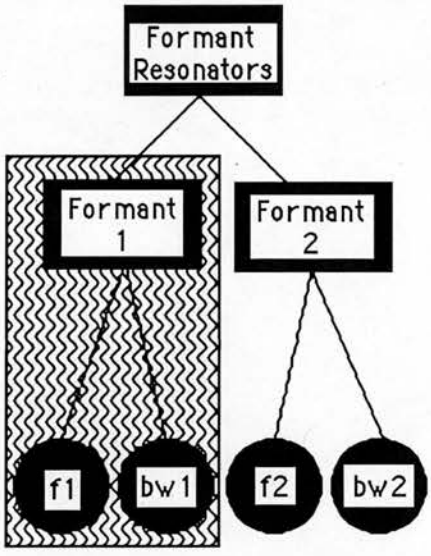


6.3.1.2 The notion of partial assemblage

Although the "layered organisation of knowledge" cognitive speculation (Chapter 4, §4.3.1) makes some degree of sense, the argument that humans always tend to access the information about a sound at its higher, more abstract level is not beyond the realm of controversy. To alleviate this problem, we should make the assembler engine flexible, so that it may also assemble single internal nodes of the schema. Besides the assemblage of a sound event (the root) there might be partial assemblages of only certain ASS nodes.

Let us assume that the example shown in Figure 6.5 has a branch of filters which constitute two formant resonators (see Chapter 4, §4.2.2). Taking as an example the node **formant 1** of Figure 6.8, it requires only its affiliated slots (namely **f1** and **bw1**) for the assemblage.

Figure 6.8: The notion of partial assemblage.



The advantage of being able to think in terms of assemblages of single ASS nodes as an alternative to the solely ASS root assemblage, is that one may now also attach non-numerical attribute values (for example, adjectives in English) to partial assemblages. For instance one could refer to the node (or attribute) **formant 1** as **low and wide** if it has the values **f1 = 250 Hz** and **bw1 = 200 Hz** (Chapter 5, §5.2.2).

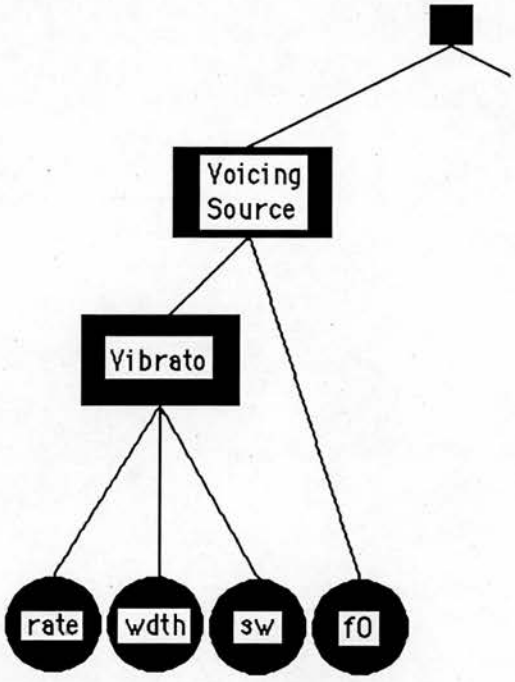
We study below how we can ask the system to produce a sound from a qualitative description (that is, from a set of attribute=value expressions) instead of actually inputting its name.

6.3.2 Referring to sounds using a user-defined vocabulary of attributes

For each component of the schema (that is, for each node of the ASS) one can define a set of possible non-numerical attribute values. Taking as an example the left branch

of the schema shown in Figure 6.5; the slots **rate, wdth** and **sw** constitute a node called **vibrato** which in turn (with the slot **f0**) form the higher level node **voicing source** (Figure 6.9). David Jaffe (1994) suggests the notion of *synthesis meta-parameter*. In our case, one could think of a node as a kind of meta-parameter.

Figure 6.9: The left branch of Figure 6.5.



One may now establish that the possible attribute values for **vibrato** are **none, uniform** and **too slow**. Each of these attribute values would then correspond to either a set of numerical values, or to a set of ranges of values within a certain interval. For example, one could define that **vibrato** is **none** if **rate = 0 Hz, wdth = 0 %** of **f0**, and **sw = 0.** The node **voicing source** is similarly defined: one could establish that **voicing source** is **steady low** if **vibrato = none** and **f0 = 55 Hz**, for example (Chapter 5, §5.5.2). This information is also represented in the knowledge base.

Hypothetically considering only this left part of the example schema, one could request the system to produce, for example, a sound by inputting the description *steady low voicing source and no vibrato* (Figure 6.10), instead of inputting the name of the sound (as in the case of the example given in Figure 6.7).

Figure 6.10: Synthesising a sound from its description.

(**1**) The user inputs a request: produce a sound with **voicing source = steady low** and **vibrato = none**.
(**2**) The assembler engine finds the attribute **voicing source = steady low** and collects the value of **f0**.
(**3**) The assembler engine finds the attribute **vibrato = none** and collects the values of **rate**, **wdth**, and **sw**.
(**4**) The assembler engine *assembles* the ASS.
(**5**) The assembler engine finally sends the slot values to the synthesis algorithm.

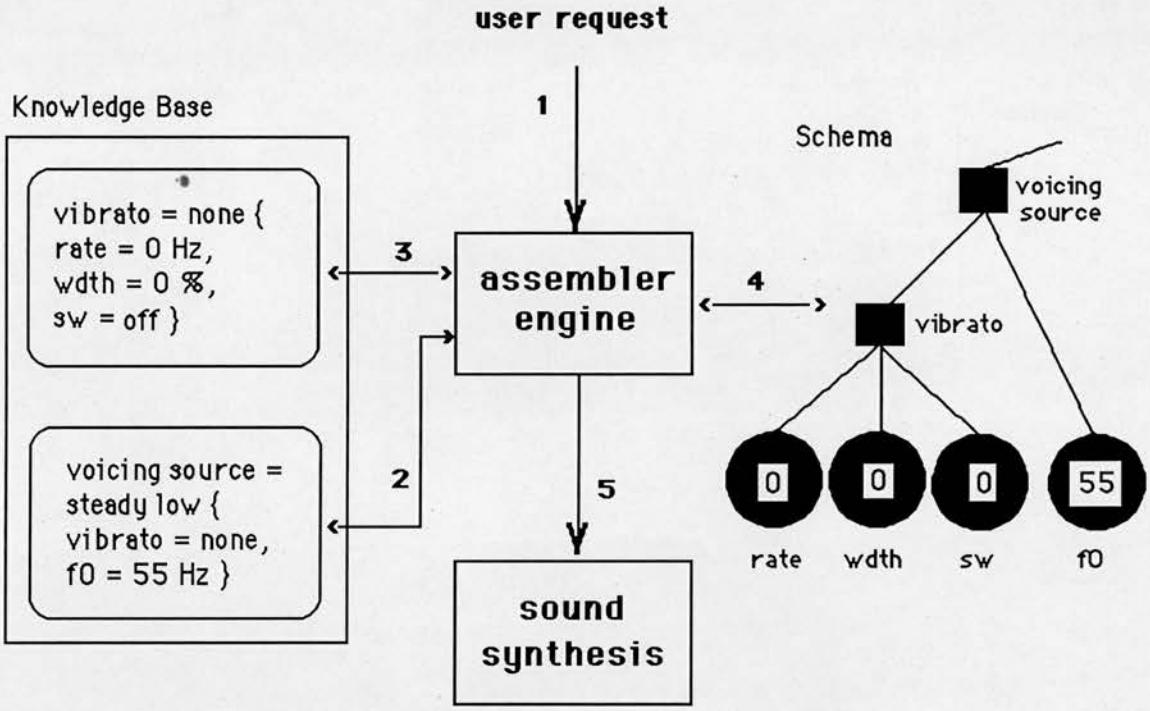
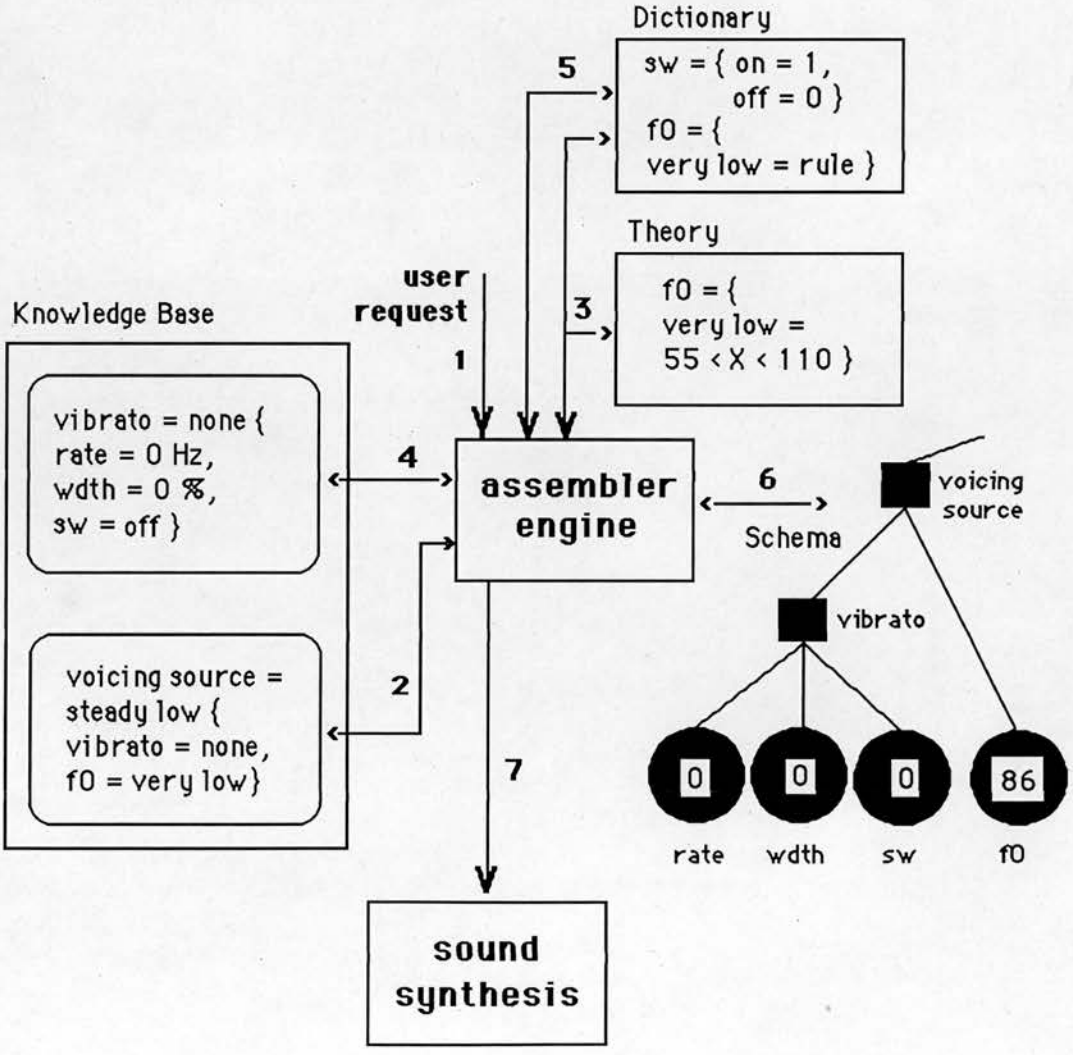


Figure 6.10 also illustrates the automatic knowledge acquisition task discussed in Chapter 4, §4.5.2 and in Chapter 5, §5.3.2. If this sound is not represented in the knowledge base, then the user is invited to label it. The system should then include this new sound in the knowledge base and record its slot values. Suppose one wished to label this sound as, for example, **gruff**; in this case the slot values would be recorded as follows:

gruff sound = {
rate = 0 Hz,
wdth = 0 %,
sw = off,
f0 = 55 Hz }

Figure 6.11: An example of how the assembler engine interprets words given to slot values instead of numbers.

(**1**) The user inputs a request: produce a **steady low sound**.
(**2**) The assembler engine finds out that the description **steady low** corresponds to the attribute **voicing source** and collects the value of **f0** and **vibrato**.
(**3**) The value of **f0** is not a number. Thus the assembler engine consults the dictionary. The dictionary tells that the meaning of the expression **very low** is calculated by a formula (or rule). The assembler engine consults the theory. The theory tells that a **very low f0** is any value between **55 Hz** and **110 Hz**. Let us suppose that the assembler engine computes the value **86 Hz** for **f0**.
(**4**) The assembler engine finds in the knowledge base the attribute value **vibrato = none** and collects the values of the slots **rate**, **wdth**, and **sw**.
(**5**) The assembler engine consults the dictionary in order to collect the value of **sw = off**.
(**6**) The assembler engine assembles the ASS.
(**7**) Finally, the assembler engine sends the slot values to the synthesis algorithm and the sound in then synthesised.



6.3.3 The role of the dictionary and of the theory modules

In the dictionary module, the user specifies a vocabulary for the slot values. This enables him to refer to slot values using words. In Figure 6.7, for example, the representation of **sound(b)** could be as follows:

```
sound(b) = {
a kind of = sound(a),
f0 = medium }
```

In this case, instead of the value **440 Hz** for the slot **f0** we used the word **medium**. The definition of this word is in this case, specified in the dictionary.

Note that in Figure 6.7 we already have specified the slot **sw = on** for **sound(a)**. In order to work out the meaning of the word **on**, the assembler engine consulted the dictionary.

In §6.2.1 above we mentioned that a slot accommodates either a sound synthesis datum, or a pointer to a formula to calculate it. These formulae are specified in the theory module.

Rather than tie a number to the "meaning" of a word of the dictionary, the user may wish to let the system calculate it, or select it from, for example, a certain range of values. One could, for instance, specify that a *very low fundamental frequency* can be any value between **55 Hz** and **110 Hz**.

In Figure 6.11. we illustrate how the assembler engine uses the dictionary and the theory modules.

6.4 The assembler engine algorithm

In this section we present the assembler engine algorithm. The algorithm is divided into four parts: one main algorithm and three sub-algorithms.

The main algorithm primarily reads the input requirement and determines which actions must be performed in order to instantiate the ASS. The sub-algorithms perform specific tasks required at various stages of the main algorithm.

Main algorithm A:

1. *The user inputs requirement*

2. *If requirement is a sound name* **SOUND**

3. *Then*

    3.1. *Performs* **sub-algorithm B.**

4. *Else*

    4.1. *If requirement contains only a list* **ATV** *of attribute values*

    4.2. *Then*

        4.2.1. *Consult induced rules:*

        4.2.2. *If there is a rule (or rules) that matches the requirement*

        4.2.3. *Then*

            4.2.3.1. *Collects the name* **SOUND** *of the matching sound (or sounds);*

            4.2.3.2. *Perform* **sub-algorithm B.**

        4.2.4. *Else*

            4.2.4.1. *Produce a list* **SLV** *of the respective slot values* **slv** *of all attribute values of* **ATV;**

            4.2.4.2. *For each* **slv** *of* **SLV** *perform* **sub-algorithm D;**

            4.2.4.3. *Perform* **sub-algorithm C.**

    4.3. *Else If requirement contains only a list* **SLV** *of slot values* **slv**

    4.4. *Then*

        4.4.1. *Perform* **sub-algorithm C.**

    4.5. *Else If requirement contains both* **ATV** *and* **SLV**

    4.6. *Then*

        4.6.1. *Collect the slot values* **slv** *of all attribute values of* **ATV** *and adds them to* **SLV;**

        4.6.2. *For each* **slv** *of* **SLV** *perform* **sub-algorithm D;**

        4.6.3. *Perform* **sub-algorithm C.**

Sub-algorithm B:

This sub-algorithm is activated either when the input requirement contains only the name of a sound itself, or when the system knows a rule which indicates a sound that fulfils the requirement.

1. Estimate which slots have to be "filled" with slv values in order to instantiate the **ASS;**

2. Consult the knowledge base in order to collect the slot values **slv** *of sound* **SOUND;**

3. If **SOUND** *does not provide all slot values necessary to instantiate the* **ASS**

4. Then

> *4.1. Collect the missing slvs from sounds of higher levels (in the knowledge base), by inheritance;*

5. For each **slv** *of* **SOUND** *perform sub-algorithm D;*

6. Instantiate the **ASS;**

7. Activate the sound synthesis algorithm.

Sub-algorithm C:

This sub-algorithm is activated when either the input requirement is either or both a list of attribute values or a list of slot values, or when there is no rule that indicating a sound which fulfils the requirement (in case of a requirement composed of attributes values only).

1. *Estimates which slots have to be "filled" with* **slv** *values in order to instantiate the ASS;*

2. **If list SLV** *of slot values does not provide all* **slvs** *necessary to instantiate the ASS*

3. *Then*

> 3.1. *Complete the list* **SLV** *by providing "default values" to missing slots;*

4. *Instantiate the ASS;*

5. *Activate the sound synthesis algorithm.*

Sub-algorithm D:

This sub-algorithm consults the dictionary and the theory modules, in order to compute the numerical value of a slot whose value has been tied to a label.

1. ***If*** **slv** ***is a word*** **W**
2. ***Then***
    2.1. ***Consult the dictionary and collect the value*** **N** ***of*** **W;**
    2.2. ***If*** **N** ***is not a number but a pointer to a rule*** **R**
    2.3. ***Then***
        2.3.1. ***Consult the theory and calculate the value of*** **N** ***according to*** **R;**
    2.4. ***Replace*** **W in SLV** ***by its respective numerical value*** **N.**

6.5 The inductive learning and the knowledge acquisition algorithms

In this section we present a brief discussion on inductive learning and examine the inductive learning and knowledge acquisition algorithms used in our case study system.

6.5.1 A brief introduction to inductive learning and algorithms used in our investigation

The target of inductive learning in our system is to induce concepts about the sounds of a training set. The use of inductive learning in our system is inspired by the "generalisation of perceptual attributes" cognitive speculation introduced in Chapter 4, §4.3.2. This speculation tells us that when we listen to several distinct sounds, we tend to make distinctions among them. To do this, we select certain attributes which we consider to be most prominent. The problem is that this selection depends upon several factors, such as context, nature of the sounds, awareness of the listener, etc. There is, in theory, a variety of combinatorial possibilities. However we do not seem to use them all, but rather to resort to a limited number of heuristics. We believe that we can simulate certain aspects of this behaviour with a computer, by using inductive learning algorithms.

Inductive learning can be either *incremental*, modifying its concepts in response to each training example, or *single trial*, forming concepts once in response to all data.

A classic example of incremental inductive learning is a program called ARCHES (Winston, 1985). ARCHES is able to learn the *structural description* of an arch from examples and counter-examples supplied by a "teacher". The examples are processed sequentially and ARCHES gradually updates its current definition of the concept being learned by enhancing either the *generality* or the *specifity* of the description of an arch. It enhances the *generality* in order to make the description match a given positive example, or the *specifity* of the description in order to prevent the description from matching a counter-example.

The Iterative Dichotimizer 3 (ID3) algorithm is a classic example of single trial inductive learning (Quinlan, 1986). ID3 induces a decision tree from a set of examples of objects of a domain; the tree *classifies* these objects *according to their attributes.* Each example of the training set is described by a number of attributes. The ID3 algorithm builds a decision tree by measuring all the attributes in terms of their effectiveness in partitioning the set of target classes; the best attribute (from an information-theory standpoint) is then elected as the root of the tree and each branch corresponds to a partition of the classifications (i.e., values of the attribute). The algorithm recurs on each branch in order to process the remaining attributes, until all branches lead to single classification leaves. A detailed study of an algorithm inspired by the ID3 technique is presented in §6.5.1.2.

There are a number of other classic inductive learning techniques largely used in AI systems. For example, Version Space Search (VSS), an incremental technique that also learns general concepts from examples and counter-examples, and Explanation-Based Learning, a single trial technique which has the ability to form explanations as to whether the examples belong to a target concept and stores these explanations as learned concepts (Luger and Stubblefield, 1989).

At the moment we are not interested in learning the structural description of sounds. This is already known by default; the ASS representation scheme explicitly defines the structure of sounds (§6.2.1). In this investigation we are interested in sound classification according to distinctive sound attributes.

We believe that we should not restrict the system to a single inductive learning technique. In principle, any technique that produces classificatory rules based upon attributional descriptions could be useful. Ideally the system should use various inductive learning algorithms in order to provide more than one classificatory possibility (that is, more than one heuristic). The ability to have more than one classificatory possibility is useful in a situation where, for example, the user inputs attribute values (as a request to produce a sound) and the system must check whether it knows a sound that matches this request (Chapter 4, §4.6.1). Therefore by having more than one classificatory rule, the system has a greater chance of finding a matching sound and indeed of finding more than one sound which satisfies the requirement. To this end, we arbitrarily selected two single trial inductive learning algorithms in our investigation: the Induction of the Shortest Concept Description (ISCD) and the Induction of Decision Trees (IDT) (Dietterich and Michalski, 1981; Bratko, 1990). We plan to add the support of incremental inductive learning techniques in the future.

The ISCD algorithm aims to induce the shortest description(s), that is, the smallest set(s) of attribute values of a sound (or class of sounds) which can differentiate it from the others in the training set. The IDT algorithm also induce classificatory rules, but not necessarily the most succinct ones. (Both algorithms are explained fully below.)

6.5.1.1 The Induction of the Shortest Concept Description (ISCD) algorithm

The Induction of the Shortest Concept Description (ISCD) algorithm induces the shortest attributional description of sounds given in a training set. (An example training set is given in Appendix VI and the instructions to make one can be found in the user's manual in Appendix VII.) The result of learning is the description of sounds (or classes of sounds) in the form of rules. The format of a rule is as follows:

```
SOUND_CLASS = [ DESCRIPTION(1),
                DESCRIPTION(2),
                ...,
                DESCRIPTION(n)]
```

where **DESCRIPTION**s are lists of attribute values in the form:

```
[ATTRIBUTE_NAME(1) = ATTRIBUTE_VALUE(1),
 ATTRIBUTE_NAME(2) = ATTRIBUTE_VALUE(2),
...
ATTRIBUTE_NAME(n) = ATTRIBUTE_VALUE(n)].
```

A sound description **SOUND_CLASS** is interpreted as follows:

(i) a sound matches with the description if it satisfies at least one of the **DESCRIPTION(n)** of a sound class

(ii) a sound satisfies a **DESCRIPTION** list of attribute values if all the attribute-value pairs in **DESCRIPTION** are as for the sound class in question.

For example, a rule for a sound class, let us say, *open vowel* might be:

```
open vowel = [      [     vibrato = normal,
                          resonators(formant) = vowel(a),
                          sex = male
                    ],
                    [     vibrato = below_normal,
                          resonators(formant) = vowel(e)
                          sex = female
                    ]
              ].
```

The interpretation of the above rule is:

> *A sound event is an open vowel if it has normal vibrato rate, its spectral envelope corresponds to the resonance of a vowel /a/, and it is a male sound, or it has a vibrato rate lower than normal, its spectral envelope corresponds to the resonance of a vowel /e/, and it is a female sound.*

The ISCD learning algorithm uses the *single trial* induction technique (Bratko, 1990) to process the examples; that is, all the input examples are processed at once. The main requirement here is that the constructed description of a sound class exactly

matches the examples belonging to the sound class. When a sound event matches a description, it is said that the description *enfolds* the sound event. Thus, the algorithm must construct a description for the given sound class, which enfolds all the examples of this sound class and no other examples. Bratko (1990) refers to such a description as being complete and correct: complete because it enfolds all the examples of the sound class and correct because it enfolds no other examples.

The algorithm works as follows:

To enfold all the examples of **SOUND_CLASS** ***in*** **SOUND_EVENTS** ***do:***

1. ***If no example in*** **SOUND_EVENTS** ***belongs to*** **SOUND_CLASS**

2. ***Then***

    2.1. **CLASS_DESCRIPTION** = [], ***i.e., an empty list***

3. ***Else***

    3.1. **CLASS_DESCRIPTION** = [**DESCRIPTION**|**DESCRIPTIONS**] ***where*** **DESCRIPTION** ***and*** **DESCRIPTIONS** ***are obtained as follows:***

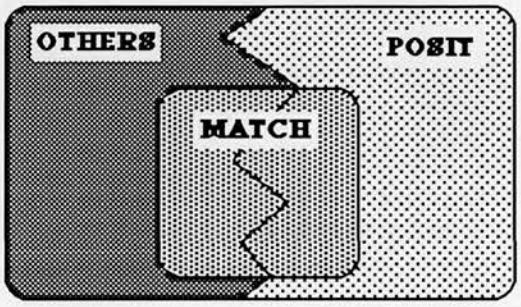
        3.1.1. ***Construct a list*** **DESCRIPTION** ***of attribute values that enfold at least one example of the desired sound class and no example of any other sound class;***

        3.1.2. ***Remove from*** **SOUND_EVENTS** ***all the objects covered by*** **DESCRIPTION** ***and enfold the remaining unenfolded sound events by*** **DESCRIPTIONS**

Each **DESCRIPTION** list is incrementally constructed. Its construction process is highly combinatorial. Each time a new attribute-value condition is added, there are almost as many alternative candidates to be added as there are attribute-value pairs. It is not immediately clear which of them is preferable. In general, we would like to enfold all the examples of the sound class being learned with as few **DESCRIPTION** lists as possible. Learning is viewed here as a search among possible descriptions with the objective of minimising the length of the concept description. We resort to a heuristic scoring function (Chapter 5, §5.4; §6.5.1), because of the high combinatorial complexity of this search. At each point, only the best-estimated attribute is added to the list, immediately disregarding all other candidates. The search is reduced to a deterministic procedure without any backtracking. The heuristics estimate (see Figure 6.12) is simple. It is based upon the assumption that a useful **DESCRIPTION**'s element should discriminate well between examples of the class being processed from the other examples. Thus it should enfold as many positive examples of the sound class as possible and as few

negative examples as possible. The heuristic scoring of an attribute value is illustrated in Figure 6.12.

Figure 6.12: The ISCD heuristic scoring function.



In Figure 6.12, **POSIT** is the set of positive examples of the sound class being learned, whereas **OTHERS** is the set of negative examples. The **MATCH** area represents the set of sound events which satisfy the attribute-value condition. The heuristic score of the attribute value is the number of **POSIT** in **MATCH**, minus the number of **OTHERS** in **MATCH**:

$$\mathbf{SCORE = |POSIT \wedge MATCH| - |OTHERS \wedge MATCH|.}$$

6.5.1.2 The Induction of Decision Trees (IDT) algorithm

The Induction of Decision Trees (IDT) algorithm also uses the single trial inductive method to learn attributional descriptions (§6.5.1.1). Here the result of the learning is represented in the form of a decision tree **DT**. Internal nodes in the tree are labelled with attributes and branches are labelled with attribute values. The leaves of the tree are labelled with sound classes. To classify a sound event, a path in the tree is traversed, starting at the root node and ending at a leaf. The IDT algorithm proceeds by searching, at each non-terminal node, for the attribute whose values provide the best discrimination among the other attributes, that is, the Most Informative Attribute (MIA). The formula for the selection of the MIA will be explained in §6.5.1.2.1.

In our work we adapted to our purposes an IDT algorithm devised by Ivan Bratko (Bratko, 1990). The algorithm works as follows (Figure 6.13):

To construct a decision tree **DT** *from a training set* **TSet** *do:*

1. *If* **TSet** *is empty then* **DT** *is a single-node tree labelled* **null**

2. *Else*

    2.1. *If all the examples in* **TSet** *belong to the same sound class* **SOUND_CLASS**

    2.2. *Then* **DT** *is a single-node tree labelled* **SOUND_CLASS**

    2.3. *Else select the most informative attribute* **MIA**

        2.3.1. *If there is no* **MIA** *to choose*

        2.3.2. *Then* **DT** *is a single-node tree with the list of the conflicting examples*

        2.3.3. *Else*

            2.3.3.1. *From the* **MIA** *obtain its attribute values* **atv(1), atv(2), ..., atv(n);**

            2.3.3.2. *Partition* **TSet** *into* **TSet(1), TSet(2), ..., TSet(n),** *according to the attribute values* **atv** *of* **MIA;**

            2.3.3.3. *Construct recursively sub-decision-trees* **ST(1), ST(2), ..., ST(n)** *for* **TSet(1), TSet(2), ..., TSet(n);**

            2.3.3.4. *The result is the tree DT whose root MIA and whose sub-decision-trees are* **ST(1), ST(2), ..., atv(2), ..., atv(n).**

Each time a new **MIA** is selected, only those attributes which have not yet been selected in previous recursions (that is, used in the upper parts of the tree) are considered.

When the available attributes are insufficient to distinguish between classes of sound examples (that is, sound examples that belong to different classes may have exactly the same attributes) then we say that these are *conflicting* examples. If the algorithm cannot find a new **MIA**, then it records a list of conflicting examples together with the number of occurrences in **TSet** of each element of the conflicting list. This information is used as a weight if a selection among them is eventually required.

Figure 6.13: Each time a new MIA is selected the algorithm constructs recursively sub-decision-trees ST for each attribute value of the MIA.

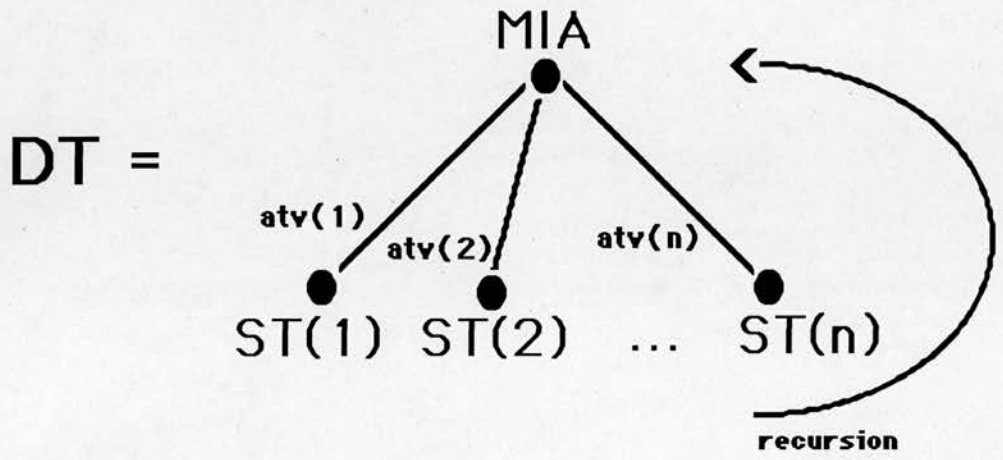
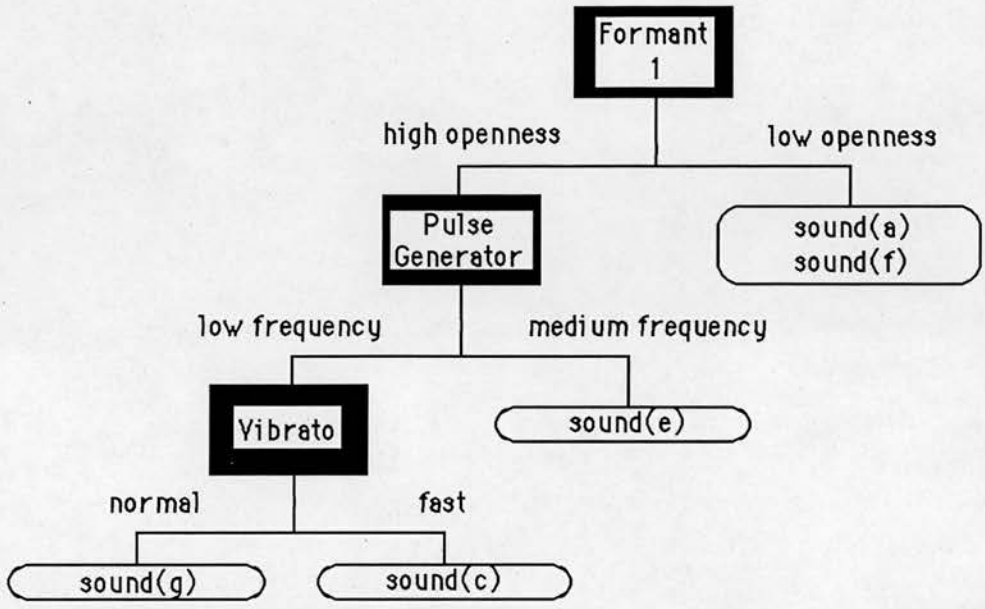


Figure 6.14 shows an example **DT** induced from an hypothetical training set. Internal nodes in the **DT** are labelled with sound attributes. The leaves of the **DT** correspond to sound classes or to an indication that there is no example to match the leaf. Branches correspond to attribute values. As we have already mentioned, in order to identify a sound, a path is traversed in the **DT**, starting at the root (top sound attribute) and ending at a leaf. One follows the branch labelled by the attribute value at each internal node. For example, a sound described by *high openness*, *low fundamental frequency* (of the pulse generator), and *fast vibrato* is classified, according to this tree (shown in Figure 6.14), as **sound(c)**.

6.5.1.2.1 The *Most Informative Attribute* (MIA) selection

The rationale behind the criterion for the selection of the most informative sound attribute (**MIA**) is that we primarily are interested in identifying the attribute that is better at discriminating between classes. It is important to observe that the **MIA** is not acoustically derived; that is, the most informative sound attribute might not correspond to what a musician would hear as being the "most informative" in practice (refer to Chapter 5, §5.4).

Figure 6.14: An example **DT** induced from an hypothetical training set.



A **MIA** is expected to divide the **TSet** into subsets, so that each subset enfolds only one sound class. This, however, is an ideal situation. The selection criterion must cope with the eventual impurity of resulting subsets. We say that each attribute of a sound has an *impurity measure*, **Imp**. Thus those attributes which minimise the impurity of the resulting subsets are preferred. **MIA** is therefore the attribute which has the smallest **Imp** value. There are a number of methods to calculate **Imp**; for example using the *entropy measure* or Bratko's *gini index* (Bratko, 1990; pp 487-488). In our prototype we used the gini index-based formula proposed by Bratko. This method is particularly suitable as it is simple to test and implement:

$$\mathbf{Gini} = \sum_{\mathbf{m}\neq\mathbf{n}} \mathbf{p(m)p(n)}$$

where:

p(m) = the probability that a sound in **TSet** is of class **m**
p(n) = the probability that a sound in **TSet** is of class **n**

For every sound in **TSet** we can use this formula to calculate the **Gini** index by adding up all the possible combinations of the probabilitites. For each attibute **ATT**, we suppose that **TSet** is partitioned into various subsets according to the values **atv** of **ATT**. The impurity measure for an **ATT** is calculated by the weighted summation of

all possible combinations of the conditional probabilities of classes, given attribute **ATT**, have value **atv:**

$$\mathbf{Imp(ATT)} = \sum_{\mathbf{atv}} \mathbf{p(atv)} \sum_{(\mathbf{m}\neq\mathbf{n})} \mathbf{p(m|atv)}\ \mathbf{p(n|atv)}$$

where

atv = values of **ATT**

p(m|atv) = conditional probability of a class **m**, given **ATT**, has value **atv**

p(n|atv) = conditional probability of a class **n**, given **ATT**, has value **atv**.

For example, assume that we wish to measure the impurity of the attribute *vibrato* and that this attribute could value *fast* and *slow*. Consider that in a training set of 7 sound examples the attribute *vibrato* appears as follows:

sound(a) = slow
sound(a) = slow
sound(a) = slow
sound(b) = slow
sound(b) = slow
sound(b) = fast
sound(c) = fast

The impurity measure of *vibrato* is calculated as follows:

```
Imp(vibrato) =    p(slow) *   (
                              p(sound(a)/slow)*p(sound(b)/slow)+
                              p(sound(a)/slow)*p(sound(c)/slow)+
                              p(sound(b)/slow)*p(sound(c)/slow)
                              ) +
                  p(fast) *   (
                              p(sound(a)/fast)*p(sound(b)/fast)+
                              p(sound(a)/fast)*p(sound(c)/fast)+
                              p(sound(b)/fast)*p(sound(c)/fast)
                              )
```

Imp(vibrato) = 5/7 * ((3/5 * 2/5)+
(3/5 * 0/5)+
(2/5 * 0/5))+
2/7 * ((0/2 * 1/2)+
(0/2 * 1/2)+
(1/2 * 1/2))

Imp(vibrato) = 5/7 * 0.24 + 2/7 * 0.25

Imp(vibrato) = 0.242

Note that **Imp(ATT)** is local. It does not reliably predict the effect of several combined sound attributes. Although it would be possible to extend its scope, in this investigation we have used the local impurity measure. Any global optimisation here would be computationally very expensive.

6.5.1.2.2 The pruning mechanism

Example training sets are often noisy (in context of information theory), that is, they may present attribute value errors and therefore errors of sound class values; this makes the learning task more difficult. When dealing with noisy training sets, it is worth abandoning the requirement that learned concept descriptions must only match positive examples. In this case, it is worth allowing the learned description to "misclassify" some of the sounds. The idea here is to allow the algorithm to ignore those examples believed to contain errors.

Inducing a **DT** from noisy data with the basic **IDT** algorithm introduced above, tends to lead to large trees with unreliable (and often unnecessary) information in them. To find a path which leads to a matching leaf in such large trees is time-consuming, as the **IDT** algorithm (besides discovering regularities) traces noises in the training data. To illustrate this, suppose that the algorithm is about to construct a subtree **ST** for **ATT = vibrato** and that the current subset **TSet** holds 80 sound examples, such that 78 of them have the attribute value **vibrato = uniform** and the remaining two have **vibrato = none**. Assuming that all 80 sound examples agree in the values of other **MIA**s already selected up to this point, it seems plausible to assume that the data **vibrato = none** is noisy. Thus it is provident to ignore these two sound examples

and return a leaf labelled with **vibrato = uniform**. Since the basic **IDT** algorithm would in this situation deliberately expand the **DT**, it is necessary to provide a mechanism to stop the expansion at this point. This mechanism is known as *tree pruning*. The tree pruning mechanism typically takes into account the number of examples in the current **TSet** the prevalence of the majority sound class at this **TSet** and to what extent an additional **MIA** selected at this point would reduce the **Imp** of the **TSet**.

Two techniques exist to implement the pruning mechanism: *forward pruning technique* and *post-pruning technique*. Forward pruning stops the **DT** expansion whilst post-pruning actually takes a complete **DT** and prunes those subtrees that are considered unreliable. We selected the latter for our investigation, due to its simplicity and straightforward functioning.

The key decision in post-pruning is whether or not to prune a **ST**. We based this decision upon classificatory error estimates of **ST**s.

In order to decide whether to prune the subtrees of a node, we must calculate its *classification error*. Let there be the subset **TSet** of sound examples that fall in a certain node **ST** and **snd** sound classes altogether in the training set. Assume that there are altogether **N** sound examples in **TSet** and the majority class in **TSet** is **CLASS**. **n** out of **N** examples in **TSet** belong to **CLASS** and **N - n** sound examples in **TSet** belong to other classes. Firstly, we pretend that the **ST** is pruned and becomes a leaf; that is, its offspring are deleted and their contents are transferred to the node. Then we calculate its local classification error **E**. Following Bratko's methodology (Bratko, 1990; pp. 492), one way in which we can calculate the classification error is to use probability. In this case we must assume that there is existing priority within the distribution of the probabilities of classes; if there is no apparent priority, then we should consider that the distribution of these probabilities is equal for all classes. The expected probability of the classification error **E** can be calculated as follows:

$$\mathbf{E(ST)} = \frac{\mathbf{(N-n)+(snd-1)}}{\mathbf{N+snd}}$$

This formula can be directly used to estimate the classification error at each leaf of the decision tree. Example (assume that there 3 different sound classes **snd**):

TSet = { sound(a), sound(a), sound(a), sound(b) }
snd = 3
N = 4
n = 3
CLASS = sound(a)

E(ST) = (1+2)/(4+3)
E(ST) = 0.43

Secondly, we calculate the *back-up classification error* of the node **ST**. For this calculation we generalise the local classification error formula in order to consider each leaf of the node **ST**. For each leaf we calculate its own local classification error. Let **ST** be a non-leaf node of a **DT** and let **ST**'s successors be **ST(1)**, **ST(2)**, etc. If the probability of branches (that is, attribute values **atv**) from **ST** to **ST(i)** is **p(i)**, then we can estimate what is called *back-up error* value of the node **ST** as follows:

$$\mathbf{Bkp(ST)} = \sum_{\mathbf{i}} \mathbf{p(i)*E(ST(i))}$$

where **p(i)** = relative frequencies of attribute values in **ST** that eventually fall into nodes **ST(i)**.

Example (assume that the **ST** of the above example has two successors):

ST = {ST(1), ST(2) }
ST(1) = { sound(a), sound(a) }
ST(2) = { sound(a), sound(b) }

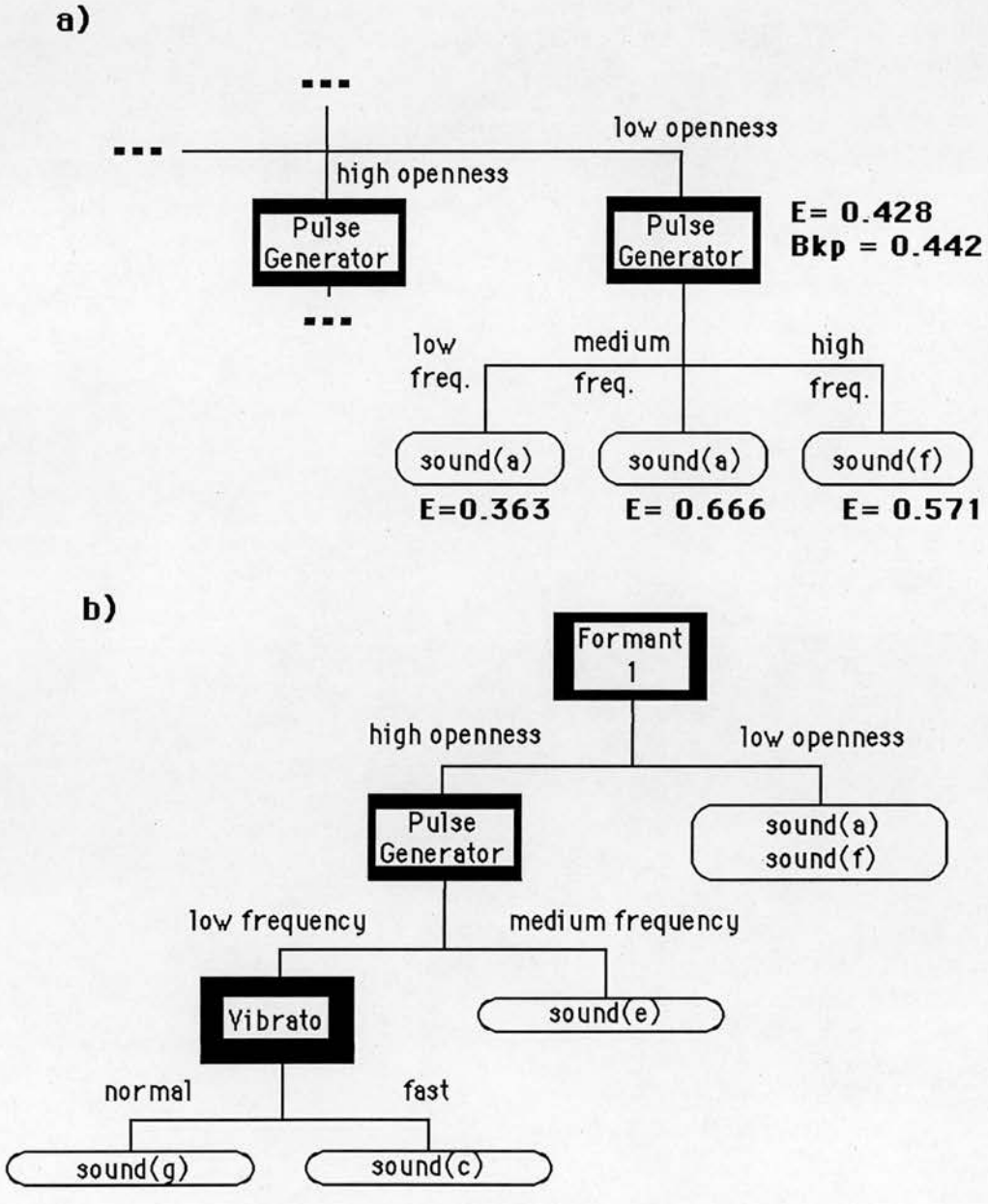
E(ST(1)) = 0.4
E(ST(2)) = 0.6

Bkp(ST) = 2/4 * 0.4 + 2/4 * 0.6
Bkp(ST) = 0.5

Finally, these two values, **E** and **Bpk**, are compared in order to decide whether or not to prune the node: if **E** is lower than **Bkp** then we prune the node's subtrees. In the example above, the node **ST** is pruned because **0.42 < 0.5**. Bratko's methodology

proved useful in this case (as for the impurity measure in §6.5.1.2.1) due to its simplicity and ease of use.

Figure 6.15: Pruning example. (a) The back-up error value of the right node *Pulse Generator* is greater than its local classification error (Bkp > E). (b) This node *Pulse Generator* is then pruned.



The pruning algorithm seeks to optimise the classification error value for a node **ST**. It works as follows:

i) starts at the bottom of the given **DT** and progresses toward the root backing up the error estimates **Bkp**
ii) if **Bkp(ST(i))** is greater than **E(ST(i))** then prune the offspring of **ST(i)**
iii) propagate upwards the lower of the two error estimate values, that is, **E(ST) = min(E(ST(i), Bkp(ST(i))**.

Thus, to prune a **ST** means to consider that what was a node with its own offspring, becomes a leaf and the contents of its offspring are transferred to this leaf. Figure 6.15 shows an example pruning of the **ST = pulse generator** of a hypothetical example **DT.** The pruning mechanism also deletes branches that lead to a "null" attribute value.

6.5.2 The automatic knowledge acquisition algorithms

The automatic knowledge acquisition task is aimed at updating the body of knowledge of the system. There are two algorithms in our system to pursue this task; one updates the system's knowledge of sounds and another updates the system's knowledge of attribute values. Both algorithms are given below.

6.5.2.1 The acquisition of knowledge of new sounds

The algorithm to update the system's knowledge of unknown sounds is very simple. An example of this task has been given in §6.3.2, Figure 6.10. It works as follows:

1. ***If the user inputs a request (for producing a sound) which contains both attribute and slot values then:***

    1.1. ***The system checks whether it knows a sound which satisfies this requirement***

    1.2. ***If it finds such a sound then:***

        1.2.1. ***Plays the existing sound***

    1.3. ***Otherwise:***

        1.3.1. ***The system creates a novel sound;***

        1.3.2. ***The user is invited to give a name to it;***

        1.3.3. ***The information about this novel sound is recorded in the knowledge base.***

6.5.2.2 The acquisition of knowledge of new attribute values

The algorithm to update the system's knowledge of unknown attribute values is similar to that introduced above. An example of this task has been given in Chapter 5, §5.3.2. It works as follows:

> 1. ***If the user inputs a request (for producing a sound) which contains slot values, then:***
>
> > 1.1. ***The system assembles the ASS and produces the sound;***
> >
> > 1.2. ***The system checks if all the slot values of this sound match with the slot values of the attribute values the system knows;***
> >
> > 1.3. ***If there is any slot value(s) which do(es) not correspond to any known attribute value, then:***
> >
> > > 1.3.1. ***Create a new attribute value;***
> > >
> > > 1.3.2. ***Asks the user to name it;***
> > >
> > > 1.3.3. ***The system records the new attribute value in the knowledge base.***

6.6 Summary

We began this chapter by proposing a system architecture embodying the knowledge level of our example study system. The proposed architecture has modules provided by the system and modules specified by the user. That is, the system provides the model for action, whilst the user specifies the information of the body of knowledge (Chapter 5, §5). We then studied the underlying concepts behind our proposed system architecture. We began this study by introducing the notion of schema. A schema is a data structure aimed at integrating and organising the body of knowledge. It has been devised to mediate a user-defined vocabulary of attributional descriptions of sounds and their respective synthesis parameters.

Inspired by the "layered organisation of knowledge" speculation (presented in Chapter 4, §4.3.1) we proposed the ASS abstract representation scheme to represent a schema, plus its inference algorithm (which is part of the model for action, as introduced in Chapter 5, §5.3) . The ASS is a representation scheme that allows us to hierarchically

represent the signal processing of an instrument, whilst providing a multi-levelled abstraction for representing sounds and sound attributes.

Sounds can be described by a number of attributes and slots using ASS. Attributes in turn may also be described by a number of attributes and slots. It is important to emphasise that sounds, attributes and slots are all labelled by the user. Having defined the ASS, the user then specifies a vocabulary (for example, by using words in English) for sounds, attributes and a dictionary of labels (also using words in English) for slot values. Once the ASS is instantiated, the system then computes the synthesis parameter values necessary to produce the sound.

The versatility of the system is that the user may then refer to a sound in various ways:

(a) by the name of the sound itself

(b) by a list of attribute=value expressions

(c) by a list of slot=value expressions

(d) by a combination of (b) and (c).

The user does not necessarily need to input an exhaustive list of attribute or slot value expressions in a requirement. The system is able to compute the missing information needed to instantiate the ASS.

We introduced the basics of machine learning and its utility for an ISSD. We focused upon a machine learning technique called inductive learning. This technique has been selected here, because we think that it is suitable to embody the "generalisation of perceptual attributes" cognitive speculation (introduced in Chapter 4, §4.3.2). Inductive learning in this system is aimed at making generalisations of "prominent" attributes of sounds, in order to make rules about them. This is useful because these rules provide means to automatically infer attributes for new sounds by analogy with existing sounds in the knowledge base. This also embodies the "identification of sound analogies" cognitive speculation (Chapter 4, §4.3.3).

We presented the role of automatic knowledge acquisition in our study ISSD. This task is aimed at allowing the system to automatically update its knowledge about

sounds and about the vocabulary for sound description through user-interaction. By acquisition of knowledge of the vocabulary, we mean acquisition of new *attribute values* and not new *attributes*. The number of attributes for sound description is delimited by the ASS; therefore, it cannot be expanded.

In Chapter 7 we study the implementations issues of the architecture.

Chapter 7

7 The engineering level

In this chapter we examine the engineering level of our case study system for sound design. At this level we study the implementation of the system's architecture. We have discussed the open-ended modules of the system's architecture (Chapter 6); that is, that there are engines and services which are provided and administered by the system, whilst there are also modules which are user-defined.

We begin by presenting how the open-ended modules (the user-specified ones) are implemented, through some examples. We then study the implementation of the engines and services provided by the system (refer to the architecture in Figure 6, Chapter 6).

We implemented our prototype ISSD with Prolog (Bratko, 1990). Prolog is a high-level programming language widely used in AI research. It has many features which facilitate the implementation of knowledge inference engines (for instance, built-in backtracking) and various procedures for data manipulation (for instance, recursion and list processing). Moreover, its declarative programming style enables us to represent knowledge in a very structured and clear way. This facilitates easy interpretation of the representation, by merely looking at the code. Thus, this programming style is beneficial for the implementation of the user specified modules, as we demonstrate below.

As an initial step to enlarge the scope and generality of this prototype ISSD (that is, for more than just subtractive synthesis) we implemented the engines and services modules as generally as possible, to allow them to work with a variety of user-specified modules. We named this software ARTIST (an acronym for Artificial Intelligence-based Synthesis Tool). We regard ARTIST as the prototype software that embodies our approach to ISSD.

7.1 Implementing the user-specified modules

In this section we study the implementation of the user-specified modules of ARTIST. The user does not necessarily need to specify all the information about sounds and attributes exhaustively, from the initial stage. The system may start with a minimum amount of user-specified information which can be expanded through user interaction. This is a particular advantage, as it allows the user to develop his specifications according to his design progress.

Default libraries of example modules are available in case the user does not wish to start from scratch. As these modules are to be user-customised, however, it may not always be particularly beneficial to exchange highly customised libraries with other users. Nevertheless, the aim is to provide the system with various default libraries for those users who do not wish to specify individual requirements.

More information on the more general implementation of these modules can be found in the user's manual in Appendix VII.

7.1.1 The instrument

In this module, the user specifies the signal processing of the instrument. A detailed discussion on the implementation of the example study instrument is given in Appendices I and II. Its fundamentals have been introduced in Chapter 4.

The instrument may be implemented by means of any suitable SWSS (Software for Sound Synthesis) package, such as CLM, (Schottstaedt, 1992; 1994), Csound (Vercoe, 1991), Mosaïc (Morrison and Waxman, 1991), ISPW Max (Puckette et al., 1992), or SOM-A (Arcela, 1994). For this prototype, the instrument has been implemented in Csound. For this particular case, we have developed a tool which translates a Prolog-generated file into a Csound score file. We are currently developing tools for the translation of the other SWSS mentioned above.

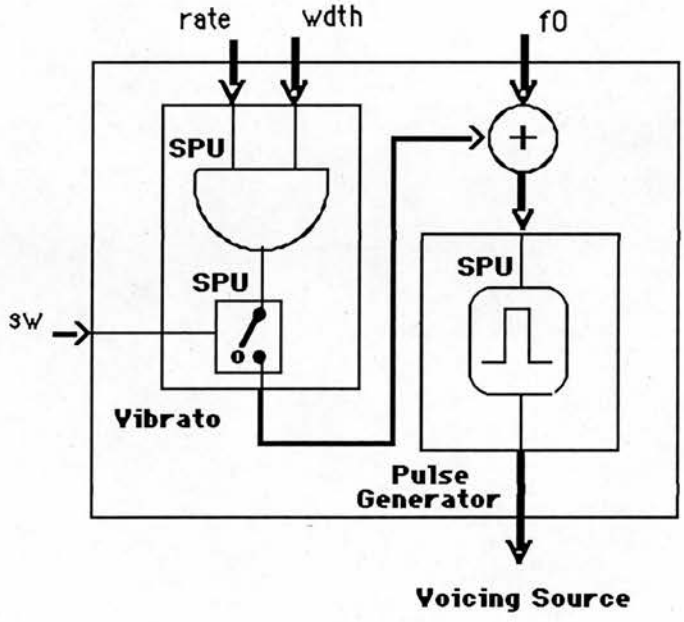
Other instrument implementation issues are discussed in the following section.

7.1.2 The schema

In the following paragraphs we study how to represent the instrument using the ASS (Chapter 6, §6.2.1).

When developing a model of a complex system (such as a synthesiser for the production of human voice-like sounds), it is helpful to divide the model into functional sub-blocks, so that its description may be given in terms of these individual sub-blocks and the mode of their connection. In our case, we have divided the signal processing block diagram (shown in Chapter 5, Figure 5.2) into two major sub-blocks, for example: *excitation* and *resonator*. The excitation sub-block in turn, is subdivided into two sub-sub-blocks: the *noise source* and the *voicing source*. Note that this is in accordance with the synthesis model we selected to implement this: the source-filter model (introduced in Chapter 4, §4.2.1.1).

Figure 7.1: The voicing source sub-block of the signal processing architecture shown in Figure 5.2, in Chapter 5.



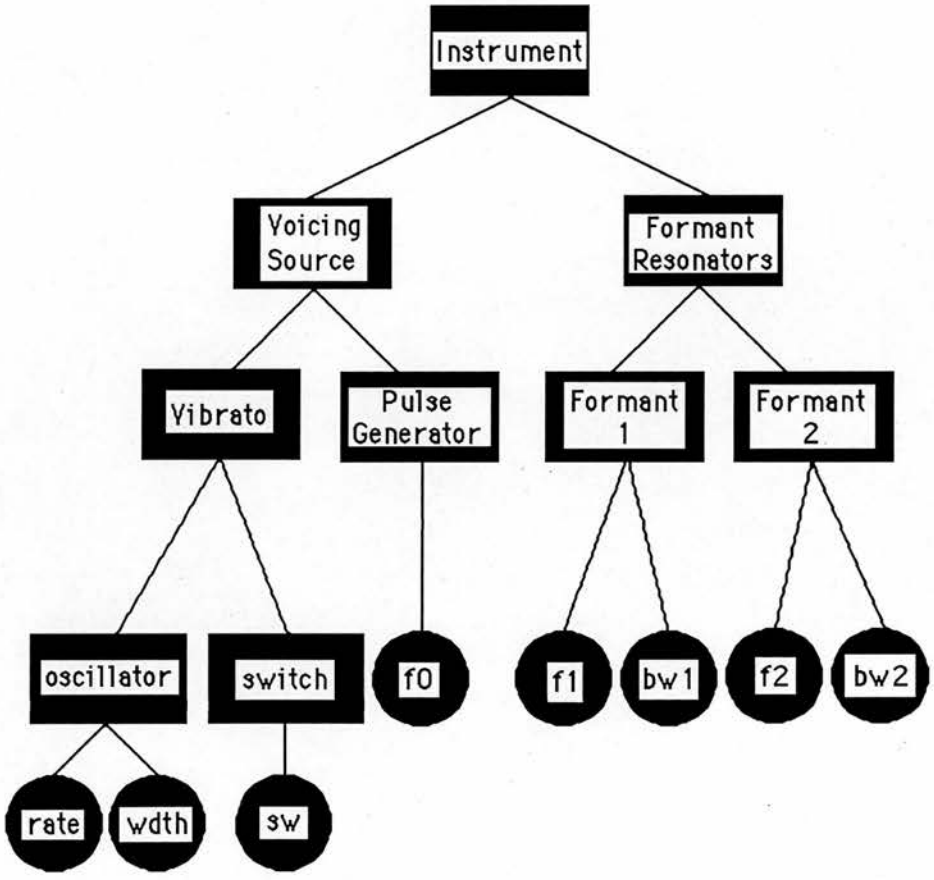
The ultimate sub-blocks of a signal processing architecture, correspond to what we call SPUs (Signal Processing Units). The *voicing source* sub-block, for example, has two sub-sub-blocks: the *vibrato* SPU and the *pulse generator* SPU (Figure 7.1). The *pulse generator* sub-block contains the *pulse generator* SPU and the *vibrato* sub-block

contains one *oscillator* SPU and a *switch* SPU (to switch the vibrato effect off, if desired).

The representation of the instrument of Figure 5.2 (Chapter 5) using the ASS, is accomplished as follows: the blocks of the signal processing architecture are represented by the nodes of the ASS and the input synthesis parameters (for instance, **sw** and **rate**) are represented by the slots.

In Figure 7.2 we illustrate a partial implementation of the schema for the instrument (shown in Figure 5.2, in Chapter 5), which involves only the *voicing source* and the *formant resonators.*

Figure 7.2: The example schema.



The leaves are slots, which correspond to sound synthesis parameters (that is, leaves correspond to values which are given to SPUs). In our case, they correspond to

Csound "pfields" (Vercoe, 1991). Slots are grouped into higher layer nodes, which are in turn grouped into yet higher layer nodes, and so on, up to the root. The links among its components are labelled *has component.* These represent the offspring relation among nodes.

The codification in Prolog (following the syntax we defined for ARTIST in the user's manual given in Appendix VII) of the above schema is as follows:

```
has_component( instrument, source(voicing) ).
has_component( instrument, resonators(formant) ).
has_component( source(voicing), vibrato ).
has_component( source(voicing), generator(pulse) ).
has_component( resonators(formant), formant(1) ).
...
...
```

The interpretation of the above clauses is as follows: an instrument has two main components: *voicing source* and *formant resonators*; a voicing source in turn has two sub-components: a *vibrato* and a *pulse generator*, and so on.

7.1.3 The knowledge base module

In *knowledge base* module, the user specifies the information of the body of knowledge (refer to Chapter 5, §5.2). This information include the names of sounds, the vocabulary for sound description and the synthesis parameters.

This module holds clusters of slot values which can be used by the assembler engine to "assemble" either a sound or an attribute (Chapter 6, §6.3.1).

We illustrate below an example of a knowledge base which contains some clusters that could be used to assemble the schema introduced above (for the syntax, refer to the user's manual in Appendix VII):

```
slot( sound_event( sound(a) ), [ rate, normal ] ).
slot( sound_event( sound(a) ), [ wdth, default ] ).
slot( sound_event( sound(a) ), [ sw, on ] ).
```

```
slot( sound_event( sound(a) ), [ f(0), low ] ).
slot( sound_event( sound(a) ), [ f(1), low]  ).
...
slot( sound_event( sound(b) ), [ a_kind_of, sound_event( sound(a) ) ] ).
slot( sound_event( sound(b) ), [ f(0), medium ] ).
...
slot( attribute( openness = high ), [ f(1), high ] ).
slot( attribute( openness = high ), [ bw(1), 78.3 ] ).
```

Note that there are two kinds of slot clusters in the knowledge base: one kind assembles a *sound event* (at the top of the example) and the other kind assembles an *attribute* (at the bottom of the example). The term "sound event" here, refers to an instantiation of the root of the schema, that is, the instrument; whereas "attribute" refers to an instantiation of a node of the schema, that is, a component of the instrument at any level (or a sound attribute). Values for the slots may be either a number or a word. In the example above, we linked the attribute **openness** to the node **formant 1** and the attribute **acuteness** to the node **formant 2** (Chapter 4, §4.4.1).

The collection of slots for the sound event **sound(b)**, is different from the other two sound events: at first sight it appears to be "incomplete". However it contains a different type of information. This new information is a link, called *a kind of*. This link associates one collection of slots with another collection. In this case, **sound(b)** inherits missing information from **sound(a)** (Chapter 6, §6.3.1.1).

It is significant that clusters of slots for a node (that is, for an attribute) cannot be incomplete. There is no inheritance mechanism for attribute values in this version of the system. The assembler engine is not yet able to identify the attribute, if a slot is missing.

In fact, there is no need to specify slots for sound events in the knowledge base. Only the slots for attributes need to be previously specified. Instead, the user may specify sounds in terms of their attribute values alone. ARTIST is able to work out the meaning of the attribute values in terms of slot values. We explain this further in our study of the specification of a training set for the *ML and KA engine* module (§7.2.2.3.1).

7.1.4 The dictionary module

In this module, the user specifies a vocabulary for the sound synthesis parameter values.

In Chapter 5, §5.3.1, we mentioned that the user can request the system to produce a sound by inputting:

(a) the name of the sound

(b) a set of attribute=value expressions

(c) a set of sound synthesis parameter values

(d) a combination of (b) and (c).

It is therefore desirable to provide a way to refer to sound synthesis parameter values by means of user-defined labels (for example, words in English) as an alternative to numbers.

The meaning of these labels in terms of synthesis values is specified in the dictionary module. With reference to the illustration given in Chapter 5, §5.3.1, one could, for example, term the values of **f0** as **{ low, high, medium }**. For each entry in the dictionary, the user specifies a list of all possible labels followed by their respective meaning for a slot. See examples below (for the syntax, refer to the user's manual in Appendix VII):

```
dict( slot( f(1) ), [   value( low,          290 ) ,
                        value( medium,       400 ),
                        value( high,         650 ) ] ).

dict( slot( f(0) ), [   value( low,          220 ),
                        value( medium,       rule( f(0), medium ) ),
                        value( high,         rule( f(0), high ) ) ] ).
```

7.1.5 The theory module

Here the user specifies a *theory* for the instrument. A theory is a set of formulae to calculate synthesis parameter values. These formulae can calculate values either based upon other parameter values, or by the random choice of a value within a certain interval.

The theory module works as an extension of the dictionary module. That is, instead of attaching a fixed number value to a label of the dictionary, the user may tie the meaning of the label to a formula (or rule) of the theory module. One could specify, for instance, that a **low** value for **f0** represents any value within a certain interval, for example, between **110 Hz** and **330 Hz**. Alternatively, the user could also specify a formula to calculate a synthesis parameter value based upon other values. One could specify, for instance, that a **medium f0** corresponds to the value of a **low f0** multiplied by two.

It is most useful to be able to specify a theory for the instrument. This provides the means to define the constraints of the sound production model. Constraints are important here for the definition of what we referred to as the *character* of the instrument (Chapter 4, §4.1). One could define, for example, a constraint for our example ISSD, which states that the value of the fundamental frequency may not exceed the value of the first formant centre frequency (if this happened, then the first formant would not resonate at all - see Chapter 4, §4.2.2).

Two kinds of formulae can be specified: a formula to calculate a slot value (that is, a synthesis parameter) based upon other slot values or a formula to calculate a slot value using of the random choice within a certain interval. See examples below (for the syntax, refer to the user's manual in Appendix VII):

```
instrument_theory( rule( f(0), medium ), F0 ):-
        get_value( f(0), low, V ),
        F0 is V * 2.

random_theory( rule( f(0), high), 680, 1080, F0 ):-
        random( 680, 1080, F0 ).
```

The first statement says that an **f(0) = medium** means the value of **f(0) = low** multiplied by two. The second statement says that **f(0) = high** is any frequency value between **680 Hz** and **1080 Hz**.

7.2 The implementation of the engines and services provided by the system

In this section we study the implementation of the engines and the services provided by the system (refer to the architecture illustrated in Chapter 6, Figure 6.1). As it is not our aim to look at detailed programming issues, we avoid listing highlights of the code in this thesis.

7.2.1 The assembler engine module

The *assembler engine* module contains the mechanism which actively uses the information in the knowledge base. That is, the mechanism that, given a requirement, both computes the slot values needed to assemble a sound and outputs the list of parameters for sound synthesis (refer to the examples given in Chapter 6, §6.3.1). For each assembled sound it outputs an *event* clause, whose atoms are synthesis parameters. The *event* clause is translated into a Csound score file and the sound is then generated by the Csound compiler.

The heart of the *assembler engine* module is a Prolog procedure called *assemble sound*. The "argument" for this procedure, is the name of the sound to be produced. It works as follows (refer also to the algorithm in Chapter 6, §6.5):

```
assemble_sound( Sound_Name ):-
        list_of_leaves( instrument, Slot_Names ),
        compute_slot_values( Slot_Names, Sound_Name, Slot_Values ),
        output_event( Slot_Values ).
```

Firstly, the procedure estimates the necessary slots for assemblage. Considering the "**instrument**" shown in Figure 7.2 the *list of leaves* procedure will return the following list:

Slot_Names = [rate, wdth, sw, f(0), f(1), bw(1), f(2), bw(2)]

Then the procedure *compute slot values*, computes the values for each slot of this list. It collects slot values, firstly within the scope of the sound **Sound_Name** and if necessary, it searches for missing slots in its predecessors (it recursively jumps to another cluster of slots linked by *a kind of*). It then consults the *dictionary* and the *theory* modules, to calculate the meaning of each slot value in terms of synthesis parameter values (only in the case where a slot value is not given as a number but as a word).

7.2.2 The user interface module

The *user interface* module provides tools of communication with the system. The user may, for example, input commands to create sounds, input training sets and consult the knowledge base. The *user interface* module also performs consistency tasks, such as checking the existence of a sound or an attribute, or corroborating attribute values.

We divide the tasks performed by the *user interface* module into three groups:

(a) tasks involving the design and manipulation of sounds

(b) tasks involving the consultation of the information of the body of knowledge of the system

(c) tasks involving machine learning.

We present below an overview of the actual operation of these tasks.

7.2.2.1 Tasks involving the design and manipulation of sounds

This group features the following tasks:

(a) playing a sound

(b) deriving a sound from another sound

(c) creating a new sound

(d) remembering previous working sessions

(e) forgetting a sound.

7.2.2.1.1 Playing a sound

This service enables the user to play a known sound. From now on the expression "known sound" refers to any sound present both in the knowledge base and in the sound files directory of the computer. ARTIST activates a procedure of the *assembler engine* module which activates a UNIX™ command (Farvis and Macintosh, 1988; Rosen et al, 1990) that plays a sound file .

7.2.2.1.2 Deriving a sound from another sound

Here the user may derive a new sound from a known sound, by informing the system of which attributes (or slots) of the source sound are to change. In order to make a request, the user must inform the system of the name of the new sound, the name of the source sound, either a set of slot values, a set of attribute values, or both.

7.2.2.1.3 Creating a new sound

This task provides ways to create a new sound from scratch, by describing which attribute values (and/or slot values) the user would like to "hear" in a sound. To make

a request, the user must input a name for the new sound and either a list of slot values, a list of attribute values, or both.

The *user interface* module consults the *induced rules* module here, in order to verify whether or not ARTIST recognises a sound whose "most prominent" attribute values match the list of attribute values input by the user as part of the request (refer to Chapter 5, §5.3.2.). If so, this new sound will inherit the missing slots from the matching sound. If not, those missing slots will be set with *default values* (the concept of "default values" is explained in the user's manual in Appendix VII, §1.2.1). In the case of more than one matching sound, the user is required to select one at will. If a match occurs and the user has not informed a list of slot values in the request, then ARTIST will play the matching sound instead of creating the new sound.

7.2.2.1.4 Remembering previous working sessions

This command loads information raised in previous usage of the system. Each time the user halts the program, ARTIST saves a file which contains all the information raised throughout the session such as, new sounds, new induced rules, and new deduced attributes. The user propels ARTIST to "remember" this information, by activating the command *remember*.

7.2.2.1.5 Forgetting a sound

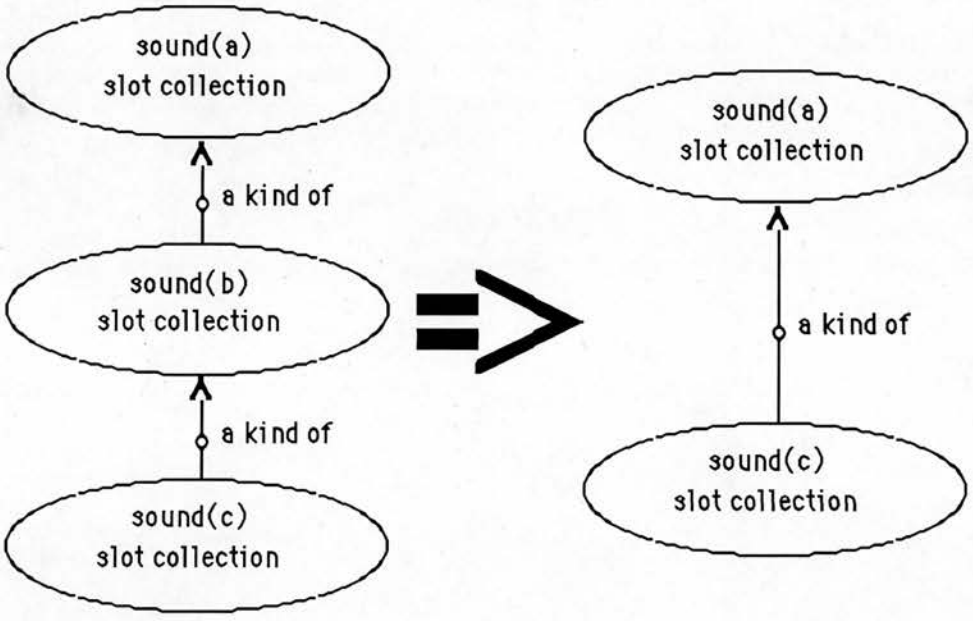
This command allows the user to remove a sound from the knowledge base and consequently from the sound files directory of the computer. If any sound inherits slots from the sound that is being removed, then the inheritance must be rearranged (Figure 7.3).

In Figure 7.3, if **sound(b)** was removed, then **sound(c)** would be adjusted in order to inherit slots from **sound(a)**. If **sound(a)** was also removed, then the missing slots of **sound(c)** would be set to default values.

The removal of a sound also conveys problems to the *induced rules* module. After removing a sound, the user should also reactivate the *machine learning engine* in order

to update the *induced rules* module. This is achieved by requesting an *introspection* in the knowledge base (dealt with in §7.2.2.3.2).

Figure 7.3: Rearranging the inheritance



7.2.2.2 Tasks involving the consultation of the information of the knowledge base

This group of tasks features the following services:

(a) displaying the name of known sounds

(b) displaying the list of attributes for sound description

(c) displaying all possible values the system knows for an attribute

(d) displaying the full set of slot values of a known sound

(e) displaying the slot values of an attribute value

(f) displaying the theory to calculate the meaning of a certain slot value

(g) displaying the list of attribute values of a sound.

7.2.2.3 Tasks involving machine learning

This group of tasks offers the following services:

(a) inputting a training set and activating the inductive learning mechanism

(b) activating the inductive learning using introspection mechanism

(c) displaying the induced rules

(d) displaying the knowledge for introspection.

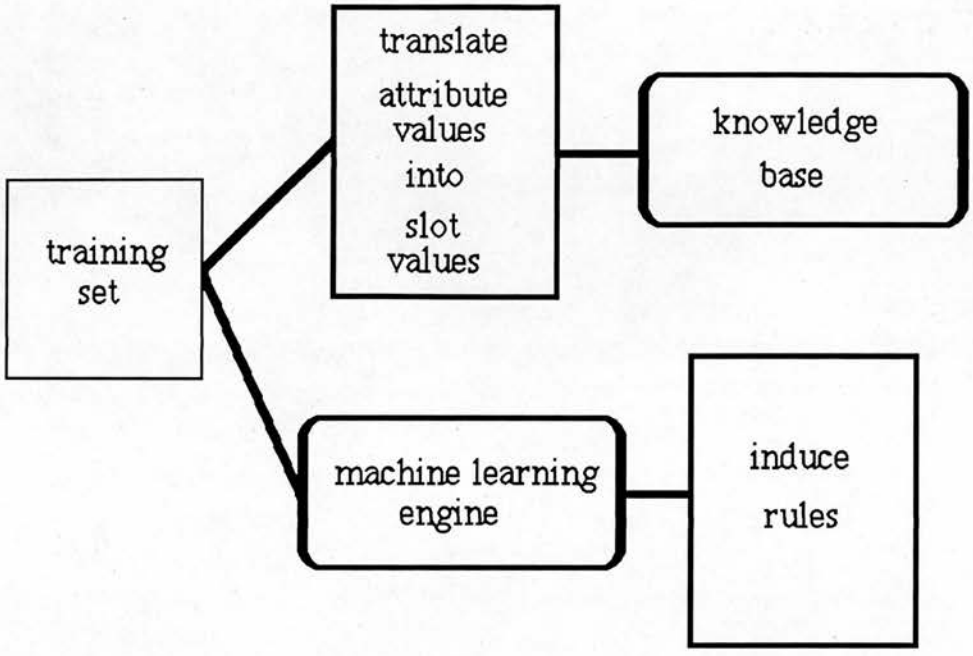
7.2.2.3.1 Inputting a training set and activating the inductive learning mechanism

Here, the user loads a training set file and activates the inductive learning mechanism. The inductive learning mechanism features two algorithms: the Induction of the Shortest Concept Description (ISCD) and the Induction of Decision Trees (IDT) (introduced in Chapter 6, §6.4). Thus, the result of inductive learning comprises two different sets of "rules". (Our use of two different algorithms here was discussed in Chapter 6, §6.5.1.)

The training set is a collection of *sound event* clauses with two arguments, the name of the sound and a complete list of attribute values. By a "complete list" we mean that all the attributes for sound description defined by the ASS should be specified for each sound of the training set. An example training set can be found in Appendix VI.

The *user interface* module performs two tasks when the user loads a training set; it gives the set to the *machine learning engine* module and it translates the set in terms of slot values, recording this information in the knowledge base (Figure 7.4).

Figure 7.4: The *user interface* module performs two tasks when a training set is loaded.



7.2.2.3.2 Activating the inductive learning using introspection mechanism

When activating the inductive learning using introspection mechanism, the system works in the same manner as the previous case (§7.2.2.3.1), but the training set is not provided by the user. Instead, ARTIST automatically builds a training set by consulting its knowledge base. In order to do this, the system employs a procedure similar to that used to display the attribute values of a certain sound (mentioned in §7.2.2.2). Instead of displaying the list of attribute values for one sound only, it records this information in a training set file for all known sounds.

7.2.2.3.3 Displaying induced rules

This task enables the user to consult the induced rules for a certain known sound. See an example of a rule in §7.2.3.1.

7.2.2.3.4 Displaying the knowledge for introspection

Here the user may consult the training set for introspection, that is, the one which the system automatically builds, by consulting its knowledge base (as mentioned in §7.2.2.3.2).

7.2.3 The Machine Learning (ML) and Knowledge Acquisition (KA) engines module

This module features two kinds of engine, namely inductive learning and automatic knowledge acquisition (their algorithms were introduced in Chapter 6, §6.5). In the following paragraphs we comment upon their targets and glance at their functioning through brief examples.

7.2.3.1 The inductive learning task

The target of inductive learning in our system, is to induce general concept descriptions of sounds from a set of examples. Our main reason for inducing rules on sounds, is that the system may then aid the user in his exploration of alternatives in the design of a certain sound (as discussed in Chapter 4, §4.5.2).

When looking for a description for a sound, **sound(c)** for example, an illustration of an induced rule could be as follows (see Figure 4.5 in Chapter 4):

> A sound is **sound(c)** if:
> it has **low fundamental frequency** and **high openness**.

Disregarding the number of attributes **sound(c)** had in the training set, (see Appendix VI for an example training set) the most relevant attributes for this sound, according to the above rule, are **fundamental frequency = low** and **openness = high**. The term *"most relevant"* here, refers to that which is most important (in the information theory sense; see Chapter 5, §5.3.2 and §5.4) to distinguish **sound(c)** from other sounds of the input training set. In this case, if the system is asked to synthesise a

sound with *high fundamental frequency* and *high openness*, then it will produce **sound(c)**.

7.2.3.2 The knowledge acquisition task

The target of knowledge acquisition in our proposed ISSD is to enable the system to update its knowledge of sounds and sound attributes, throughout user interaction. When updating the knowledge of attributes, for example, this task could be illustrated as follows: the input requirement for producing a sound may contain either attribute value statements (for example, **vibrato = fast**), slot value statements (for example, **f(0) = 55 Hz**), or both. An attribute value often depends upon the values of several synthesis parameters. In Chapter 5, Figure 5.3, for example, the attribute **vibrato** depends upon the values of the **rate** and **wdth** parameters. The system should be able to identify which synthesis parameter values correspond to certain attribute values. The task of the program here, is to infer whether or not input slot values in a requirement match known attribute values. If there is no match, then the system automatically adds this new information to the knowledge base and asks the user to name it. Suppose that the system knows three values for the attribute **vibrato** (see Appendix I, §1.1.1, for an explanation of vibrato) as follows:

vibrato = none if { **rate = 0 Hz,**
wdth = 0 % of fundamental frequency }

vibrato = uniform if { **rate = 5.2 Hz,**
wdth = 3 % of fundamental frequency }

vibrato = too slow if { **rate = 3.6. Hz,**
wdth = 3 % of fundamental frequency }

If the user requires a sound with **rate = 12 Hz**, for example, the system should synthesise it and deduce that it is not aware of any attribute value for vibrato, whose **rate** equals **12 Hz**. In this case, the system has to add this new information to its knowledge base: it must compute the missing synthesis parameter values needed to create this new attribute value (in this case, **wdth**) and must also ask the user to name it. Let us say, for example, that the user wishes to call it **tremolo**. The system, in this case, should add the following information in its knowledge base:

vibrato = tremolo if { **rate = 12 Hz,**
wdth = 3 % of fundamental frequency }

7.3 Summary

In this chapter we have presented the engineering level of our case study system. We studied the implementation of the architecture, discussed at the symbolic level in Chapter 6. We implemented this architecture in Prolog and named it ARTIST. ARTIST is a piece of software which endeavours to embody our approach to ISSD.

ARTIST provides several AI-based services, aimed to help the user in sound design, but it remains open-ended regarding the specification of the signal processing of the instrument and the definition of the vocabulary for sound description.

We began this chapter by demonstrating how the user could implement the open-ended modules. We have also focused upon the implementation and functioning of the main built-in wheels and excels of the system.

In Appendix V we provide a sample operation of ARTIST: it demonstrates how to design three sound examples using this system.

Chapter 8

8 Evaluation, conclusion and further work

In this chapter we aim to:

(a) attempt to evaluate the work

(b) review our approach to ISSD design

(c) draw some conclusions

(d) suggest further work.

In this thesis we have proposed an AI approach for the design of ISSD systems. By an ISSD system, we mean a system which allows for the design of sounds, thought of in terms of qualitative descriptions (by using words in English, for example). The system should provide intelligent aid for the exploration of the capabilities of a synthesis algorithm.

To test our approach, we devised a prototype ISSD, which implements a specific case study. Although for this case study we adopted a particular synthesis model (subtractive synthesis of formants), we carefully made the conjectures and abstractions as independent as possible of this specific model.

We began with the assumption that sound design is an explicitly knowledge-based kind of intelligent behaviour. We then elucidated the behaviour for our case study system, by suggesting some capabilities believed to be desirable in an ISSD. We conjectured the kind of knowledge the system should possess in order to exhibit these capabilities, studied how this knowledge could possibly be handled by the computer and proposed a system architecture. The next stage of our investigation involved the implementation of the prototype: an "intelligent" system for the design of "formants", named ARTIST.

ARTIST provides two levels of operation: the *instrument design level* and the *sound design level.* At the instrument design level, ARTIST offers a method to implement a synthesis algorithm (or instrument) and to specify the vocabulary of sound descriptors. At the sound design level, ARTIST offers a series of facilities to design sounds. At this level, the user works with a particular instrument and vocabulary, either provided by the system (several libraries could be available) or defined by the user at the instrument design level.

In the following section we evaluate ARTIST in the context of the interplay between the instrument and sound design levels. Consequently, we consider the implications of our AI approach for the development of ISSDs. We present an overview of our approach and summarise its strengths and limitations. We end the thesis by suggesting what we think could be done in order to enhance this work.

8.1 Evaluation of the case study system

In Chapter 2, §2.5, we discussed some evaluation issues and proposed a method to evaluate the system. In the following paragraphs we attempt to evaluate our case study system in order to settle the questions: "Does it fulfil the desirable capabilities of an ISSD?" and "What can we learn from this research and from its underlying concepts?". We believe that the discussion of these will enable us to evaluate whether our approach to ISSD can possibly lead to a better sound design system for the composer's desktop.

8.1.1 The fulfilment of the desirable capabilities

In this section, we attempt to assess the sound design level of our case study system, based upon the desirable capabilities of an ISSD (Chapter 2 §2.3).

Four people were invited to test the system (three composers of the Music Faculty and one researcher of the Artificial Intelligence Lab of Paris University VI). We asked them to comment upon it. We provided a user's manual (Appendix VII) and some operational examples in the form of a tutorial (Appendix V). Apart from the French researcher, our guests had a demonstration on how to get started with ARTIST. The author also explained some parts of the tutorial. Our guests were asked to go through

the example operation on their own and to explore the system at will. We did not control how many times they tried the system, or how long each section took. Four weeks later they were asked to report their impressions in informal interviews. Their comments, added to our own criticism, resulted in the following evaluation.

8.1.1.1 Response to intuitive sound descriptions

8.1.1.1.1 Our guests' comments

Not all of our guest users were familiar with the fundamentals of formant synthesis. Consequently, the proposed vocabulary was unfamiliar to them. Nonetheless, after getting accustomed to it, most of our guests reported that the system responded satisfactorily.

One of our guests added that the system could also be a good vehicle to introduce formant synthesis. He said that, although the vocabulary had not been specified by himself, he found that the system facilitated the study of the capabilities of the implemented synthesis technique: it is far more intuitive than low-level parameter settings. Our proposed system therefore also has potential for another class of software we had not previously considered, that is, tutorial systems.

Another of our guests expressed that he had expected a fully working system with a larger scope, so that it could be effectively used to generate all the sounds he could imagine for the composition of an entire piece of music, in only a few hours. One lesson to be learned here, is that when a researcher presents a prototype system to potential users, he must clearly state that it is a prototype and not an off-the-shelf product, that can solve all their compositional problems. It seems that the "intelligent system" AI jargon should be used more reservedly in such situations, in order to avoid unrealistic expectations.

8.1.1.1.2 Our own criticism

To answer the question "Does the system respond to intuitive sound descriptions?", we have found that it does respond on condition that (a) the system is set up with a good vocabulary for attributes, slots and their values and (b) this vocabulary is kept

coherent by the user. By maintaining the coherence of the vocabulary, we mean creating labels that make sense. A label makes sense if the sound quality it should refer to corresponds to that which we hear. This implies that if one were to create the attribute "vibrato", for example, a vibrato generator must be available. Furthermore, the labels for the attribute values for vibrato should be consistent. For example, if one establishes that the vibrato of a sound is "yellow", then a colour metaphor should also be used to refer to other types of vibrato.

We have achieved a good response to intuitive sound descriptions in our case study system (§8.1.1.1.1). We illustrate this in the following paragraphs, by examining the first design example of the tutorial given in Appendix V, §4: the design of the *distorted vowel in a low register.*

There are three key terms in the description "distorted vowel in a low register": "distorted", "vowel", and "low register". Each of these expresses the quality of an imagined sound. We study how the system responded to this description.

We suggested that the *distortion* effect can be achieved by adding noise to a vowel sound and by exaggerating its vibrato (Appendix V, §4.2). The noise effect is related to the attribute "roughness" of our instrument whereas vibrato is related to the attribute "vibrato". As for the descriptive term "vowel", ARTIST was already aware of several vowel sounds. Thus we did not have to specify the attributes for a vowel quality; we simply asked ARTIST to collect these attributes from its knowledge base (by using the "deriving a sound from another sound" command (Appendix VII, §2.1.1)). We had only to select a vowel. In this example we have selected the vowel /a/ (as in the word "bat" in English). Finally, the *low register* effect is related to the attribute "register" of our instrument. From the following requirement, the system produced a *distorted vowel in a low register*:

```
{   original sound = vowel_a,
    register = low,
    vibrato = distortion(typical),
    roughness = noisy        }
```

8.1.1.2 User configuration

This requisite is only implicitly related to the *sound design level.* It is in fact explicitly related to the *instrument design level.* It is the instrument (consequently the vocabulary) and not the algorithms of the sound design level which are configured here.

8.1.1.2.1 Our guests' comments

Unfortunately, none of our guests adventured to configure the system, mainly because the sub-routine which translates the output of the Prolog program to a Csound score file is not yet fully implemented.

Despite this, one of our guests did experiment with customisation. He was dissatisfied with the means we provided to refer to the envelope of the sounds he wished to design (the envelope is explained in Appendix I, §1.3). Originally, the system only provided labels to refer to an envelope as a whole, whilst he wished to refer to the sections of the envelope, namely: attack, decay, sustain, and release. This feature was customised in a few minutes with the help of the author.

8.1.1.2.2 Our own criticism

Although the system has not been configured to synthesis models, other than subtractive synthesis of formants, we suggest that ARTIST fulfils the user configuration requirement. ARTIST's architecture features open ended modules which are intended to be entirely specified by the user (Chapter 6, §6.1). In these modules the user implements the domain of the system: the synthesis algorithm and the vocabulary of sound descriptors. The user, however, does need some expertise to be able to configure the system properly.

Our own criticism indicates certain constraints and difficulties concerning the user configuration of our system.

Firstly, one must consider that there are certain synthesis techniques which suit our approach better than others. The preferable techniques are those which allow one to use their acoustic counterpart as a metaphor to think about sounds. We cite, for example, subtractive synthesis and physical modelling techniques as two suitable techniques (§8.3). Other techniques, such as additive synthesis and distortion, may demand other metaphors, which are beyond the scope of our research so far. This issue certainly deserves more attention in future work.

Secondly, we spent a great amount of research effort in order to work out a coherent vocabulary to describe those sounds which our instrument could produce. Devising a relevant vocabulary is a task which demands some erudition; most of the difficulties we encountered here stemmed from finding a suitable point of departure for study. The specification of a coherent vocabulary is a very important issue, as the efficiency of our system relies upon its configuration. So far we have only introduced an example study, suggesting a method to devise a suitable instrument plus a coherent vocabulary and have indicated some related bibliographical information (Chapter 4, §4.4). It would however be valuable to devise more efficient and less obscure methods.

Finally, note that it is the domain of the system which is customised, not its functioning. The user has no access to the algorithms of the system, apart from writing certain kinds of procedures in the *theory* module (Chapter 7, §7.1.5).

Further research is required in order to reduce the amount of expertise needed to customise our system and perhaps, more research should be done to check whether the customisation of certain functional aspects would enhance ARTIST.

8.1.1.3 Intelligent exploratory aid

8.1.1.3.1 Our guests' comments

The prevailing opinion was that the facilities provided by the *user interface* module (Chapter 6, Figure 6.1), such as the ability to consult various kinds of information on the knowledge base (Chapter 7, §7.2.2.2), enhance the exploratory aid aspect of the system. The ability to design a sound by prototyping partial solutions gradually, was also welcomed (the desired sound can be achieved by changing the attributes of

sounds either already known by the system, or created from scratch, until the optimal solution is reached).

None of our guests, however, were particularly impressed by the interface we devised (by "interface" we mean the way the user interacts with the system: we do not refer here to the *user interface* module of the architecture). The ability to communicate with the system in a quasi-natural language-like fashion is desirable and very promising. Our guests, however, were not completely satisfied with the use of the command line to input their requests. It was suggested that commands and other input information, could be available in various sets of windows containing menus and buttons, so that the user could handle them by using mouse selection and drag-and-drop facilities.

We believe that the interfacing problem does not really affect the evaluation of the exploratory mechanism provided by our prototype. We agree, however, that a bold desktop, mouse-based interface would be very helpful. This is an engineering aspect that deserves more attention in future implementations (§8.3). This problem is being partially alleviated by a new interface we have developed subsequently using Emacs (Emacs is an screen editor package (Rosen et al., 1990)).

8.1.1.3.2 Our own criticism

Although the term "intelligent" can be debated here, we propose that ARTIST provides intelligent aid for exploration. The fact that the user can request the production of sounds using "incomplete" descriptions and that the system is then able to work out the necessary synthesis parameters, is evidence of the system's ability to aid exploration. In addition, ARTIST also features the ability to induce rules about sounds either represented in its knowledge base, or given in an input training set. Consequently, we can regard ARTIST as an "intelligent" system, as it has the ability to fill in missing slot values in the synthesis algorithms by utilising rules which provide the maximum amount of coherent data available.

The second sound design example of the tutorial given in Appendix V, §4.3, illustrates how ARTIST uses its induced rules in order to provide intelligent exploratory aid. In this example, we wanted to produce a sound with medium acuteness. As we had only a vague idea about this sound, however, we did not know which other attributes could also be input in the requirement. We thought that perhaps ARTIST could help us.

The input requirement for this example was simply: **acuteness = medium** (note that the acuteness effect is related to the attribute "acuteness" of our instrument). ARTIST knew a rule which suggests that a vowel /a/ (as in the word "bat" in English) is a good example of a medium acuteness sound and therefore produced this sound as a suggestion. ARTIST is able to make an "intelligent guess", by suggesting a sound; this may or may not satisfy the user's requirement. This outcome can serve, however, as a starting point for the design of the optimal sound, by a gradual specification of its attributes.

8.1.1.4 Ability to learn

8.1.1.4.1 Our guests' comments

This feature was not specifically commented upon by our guest users, as they were not required to customise the system. They were provided with a case study which had previously induced rules.

Congratulatory comments were made, regarding the knowledge acquisition of new attribute values. Not all of our guests felt comfortable, however, in labelling newly deduced attribute values: it is not yet clear why. We suggest that this could be caused by the fact that they did not create the vocabulary and in this case it can be difficult to label terms coherently. It was suggested that perhaps the system could prescribe labels for new attribute values; for example, suppose that the system knows the following values for the attribute "frequency": **low** and **high**. If the system deduces a new frequency value which falls between these two known values, ARTIST might prescribe the label **medium** for it. In such cases, ARTIST should also be able to cope with finer descriptions, such as **more high**, **less low**, **much more low**.

8.1.1.4.2 Our own criticism

We have seen that ARTIST has the ability to learn. It not only has the ability to update its knowledge base automatically, but is also able to induce rules for sounds. Moreover, it has the ability to use more than one heuristic to induce these rules. This

ability enriches the exploratory aid capability, by offering a multiple classificatory view of its knowledge.

One problem surprisingly went unnoticed by our guests: the fact that induced rules might not always be compatible with our perceptual judgement (Chapter 5, §5.4). Further research into machine learning demands more attention, to check whether techniques other than inductive learning, could help to alleviate this problem. It is likely that neural networks, for example, might deserve more credit in ISSD design (§8.3).

As far as the knowledge acquisition feature is concerned, ARTIST does not actually expand its knowledge of the *attributes* that could be used to describe a sound. Rather, it only acquires knowledge of new *attribute values*, that is, variances (or "instantiations") of the existing attributes. For example, the label **fundamental frequency** is an attribute, whereas **high, medium** and **low** could be its potential attribute values. The label **extremely high**, could well be an attribute value which the system might eventually "learn". Further research is needed to devise a means for the acquisition of new attributes for sound description, in addition to the acquisition of attribute values.

8.1.1.5 Uncertainty factor

8.1.1.5.1 Our guests' comments

Our guests did not make explicit comments on the uncertainty factor, but they were implicit in their comments on the exploratory aid. They reported that there were a few occasions where they deliberately required ARTIST to synthesise something from scratch. In those cases, the outcome often suggested the next design steps. This is indeed one of the provisions we intended to offer in our system.

One of our guests, who is enthusiastic about uncertainty in music, has put forward the idea that he wished to specify rules (for the *theory* module (Chapter 7, §7.1.5)) for random selection of values within larger intervals, so that the randomness would carry a significant amount of audible affect. This idea merits further experimentation.

8.1.1.5.2 Our own criticism

ARTIST appears to fulfil the capability to provide uncertainty. Further research is necessary, however, to improve this feature.

The outcome of an incomplete requirement may be fairly unpredictable, but this depends upon the degree of "incompleteness" of the requirement. A requirement may be incomplete, but if the information it contains is "very strong", then the outcome would in fact be highly predictable. A requirement containing only the statement **fundamental frequency = low**, for example, will obviously result in a low pitched sound: it is a fairly incomplete request but the outcome is predictable to some degree.

8.1.2 Other things we can learn from this work

In addition to the various matters studied above, this case study system teaches some other significant aspects.

We have learned that it is possible to devise a user-customised sound design system that responds to qualitative sound descriptions. However, the key issue for this research is to provide maximum generality without losing coherence.

Coherence refers to the degree of closure amongst the vocabulary for sound description understood by the system, the synthesis algorithm which responds to this vocabulary and the sounds produced by this algorithm in response to the vocabulary. Generality refers to the flexibility of the system with regard to the definition of this vocabulary and the specification of this synthesis algorithm.

One could devise a system with a good degree of coherence, by tying it to a specific synthesis technique and to a circumscribed vocabulary for sound description. In this case, all the mappings are established beforehand by its programmer. This vocabulary therefore reflects the way in which the programmer understands a particular sound domain and tends to be fairly narrow indeed. In this case, the programmer would certainly specify everything in a highly disciplined way, so that the labels of the

vocabulary would carry significant perceptual information to the synthesis algorithm. Thus, the degree of coherence between this vocabulary and the sounds it describes tends to be very high, but the generality of the system is kept low.

Alternatively, one could devise a system with good degree of generality, by providing ways for the user to create a customised vocabulary and design his own synthesis algorithm. In this case, the system should offer a formalism to represent this information, facilitating the specification of the mappings within it. The degree of coherence between this vocabulary and the sounds it seeks to describe is not, however, guaranteed by the system. It is dependent upon the skills of the person specifying this information. In this case, the system could yield a high degree of generality but it would not guarantee its coherence.

The designers of SeaWave, for example, adopted the former approach (Chapter 3, §3.4). We adopted the latter approach for our investigation, as we think that a good degree of coherence may be achieved anyway, through disciplined customisation. Therefore, we directed our efforts towards the provision of generality.

8.2 An overview of our AI approach to ISSD design

Our AI approach to ISSD design enabled us to devise suitable abstract structures to represent and integrate knowledge, plus the algorithms for knowledge inference, machine learning and knowledge acquisition. This approach played an important role for the provision of generality to our case study system: most of the AI features we devised would not be feasible without this provision (for example, the system would not be able to update its knowledge using the user's own terms, if the system was tied to a circumscribed vocabulary for sound description).

In the following paragraphs we review our approach. We attempt to detach it from our case study so that it can be proposed as a generic approach to ISSD design.

We attempted to propose a generic approach for two reasons. It can serve as a generic "methodology" to study ISSD design; that is, given a synthesis device controlled by numerical streams, we suggest a means to control it via a vocabulary of higher level sound descriptors (for example, using words in English). Alternatively, it can serve as an instrument for the advancement of our own project; it can facilitate the

identification of those aspects or steps which need more attention (for instance, the "knowledge acquisition of new attributes" problem (§8.1.1.4.1) certainly demands more research at the symbolic level).

8.2.1 The knowledge level

8.2.1.1 The specification of the behaviour of the system

At this initial stage one should begin to sketch the behaviour of the system. For this sketch one must consider three factors:

(a) the goal of the system: to produce a sound from its attributional description

(b) the synthesis model at hand: we took the subtractive formant synthesis model. Other models, however, may be used. Those models which allow one to think about sounds by making analogies with their acoustic counterpart are preferred (for example, synthesis by physical modelling)

(c) the body of knowledge the system needs to accomplish its goal: the synthesis parameters, the vocabulary for sound description and a method to connect them. To have a greater understanding of the theoretical background of the body of knowledge we considered other musicological fields (such as psychoacoustics).

8.2.1.2 The detachment of information

The next stage involves detaching the information from the selected sound producing model. The initial sketch of the behaviour of the system (§8.2.1.1) should suggest the body of knowledge which is required in order to devise the system. At this stage one should devise which information these structures should possess and the model for action to process this.

One must specify at this stage the synthesis parameters of the instrument, study their role (for example, Appendices II and III) and define a vocabulary. We proposed a method to accomplish this. This method suggests that the definition of the signal processing architecture of the instrument, plays an important role in the specification

of the vocabulary. It is important to design a signal processing architecture whose modules perform specific tasks, so that one can relate them to perceptual attributes. We distinguish two types of terms of the vocabulary for sound description: *attribute* and *attribute values*. "Attribute" is the label which one ties to a module of the signal processing architecture. An attribute can be thought of as a "meta-synthesis parameter", because its value depends upon the values of the synthesis parameters of the module. "Attribute values" are the labels one relates to the outputs of a module (that is, "values" of an attribute). We also suggested that vectors are useful to ascribe attribute values. Each attribute may be made to correspond to a vector. Sub-vectors in turn may correspond to different perceptual stages, according to the values of its synthesis parameters.

We also propose the definition of a *dictionary* of terms to refer to synthesis parameter values, so that the user may also define labels for single synthesis parameters.

Finally, one must define a model for action; that is, how the system should process this information in order to accomplish the goal (Chapter 5, §5.3).

8.2.2 The symbolic level

8.2.2.1 The definition of the abstract structures for knowledge representation

The abstract structures for knowledge representation are devised at this stage. Here one devises a representation scheme and defines the structure of the knowledge base. We proposed the ASS schema for this purpose (Chapter 6, §6.2.1). The ASS schema is useful because it allows hierarchical organisation of knowledge and features an inheritance mechanism (Chapter 6, 6.3.1.1).

8.2.2.2 The definition of the algorithms

The algorithms which access and use the information recorded in the knowledge base, that is, the algorithms of the model for action, are defined at this stage. In our case, we have devised the algorithms for knowledge inference, inductive learning and knowledge acquisition.

8.2.2.3. The definition of the system architecture

After the definition of the abstract structures and the algorithms, one can devise an architecture to embody them.

We suggested that this architecture must feature open ended modules. That is, there must be both operational modules (featuring the algorithms and several additional interfacing tools) and open-ended modules (which contain user-specified information).

8.2.3 The engineering level

Finally, the architecture is implemented using appropriate software and hardware. It is important to select a suitable programming language for this purpose.

We suggested the use of Prolog (Bratko, 1990). Prolog features several built-in procedures (for example, backtracking, pattern matching and list processing) which facilitate the implementation of the knowledge representation scheme and the algorithms for knowledge inference. Other languages such as Smalltalk and Lisp are also suitable (Luger and Stubblefield, 1989).

8.3 Conclusion

In this thesis we presented a completely new approach to sound design systems: an AI-based approach.

Sound design using current sound synthesis packages is time-consuming and engages the composer in a considerable amount of non-musical tasks; such as low-level parameter settings for each single sound. Sound design is a complex task that demands a large amount of knowledge; current sound synthesis system packages do not help the user to manage this knowledge. Our approach combines sound synthesis with AI in order to alleviate this problem.

In order to test our approach we devised a case study ISSD which works by:

a) responding to user-customised sound descriptors in English
b) supporting the user in the exploration of the capabilitites of a sound synthesis algorithm
c) automatically expanding its knowledge about sounds through user interaction.

8.3.1 The strengths of our approach

On the whole, the strengths of our case study ISSD could be summarised as follows.

We do not tie the system either to a particular sound synthesis technique, or to a circumscribed vocabulary for sound description. These are entirely user-defined. The link between the vocabulary for sound description and synthesis parameters is forged by the relation between the implementation of the instrument and the definition of the labels defined by the user for its reference. It is up to the user to implement a suitable instrument, so that it can support the definition of a coherent vocabulary to describe the sounds it can produce.

The user is provided with a means of access to the instrument at various levels, ranging from direct access to numerical settings, to higher level descriptions of the instrument's behaviour (for example, an attribute might refer to a group of parameters of a certain "module" of the instrument, functioning as a kind of meta-parameter). Moreover, if the user inputs numerical settings, the system is able to deduce the possible significance of these numbers in terms of higher level descriptions. This is done by means of an automatic knowledge acquisition engine that has the ability to update the system's knowledge of sounds and attribute values, through user interaction.

Finally, we offer powerful knowledge engineering techniques which enhance the performance of the system (for example, "intelligent" aid for exploration). Furthermore, the system can assist the user in concept formation (for example, the user may consult the generalisations made by the machine).

8.3.2 The limitations of our approach

We can summarise the limitations of our case study ISSD as follows.

The definition of a coherent vocabulary for sound description is not a straightforward task. We provided only a preliminary study, using an example biased towards a certain class of synthesis methods, which allow use of its acoustic counterpart as a metaphor to think of sounds. It is still unclear how one could accomplish these independently of the synthesis model at hand.

We do not provide support to the process of exploration when designing a sound. All exploratory steps are carried out manually. The power of the computer could also be used to assist the user in this task. For instance, the system could keep record of intermediate stages of design so that they could be recalled in future design processes.

8.4 Further work

In addition to all the improvements suggested during the evaluation, we suggest some further work needed to be done in order to enhance this research.

We should find a way to build a closer link between the labels one ties to an ISSD and the perceptual vocabulary which he would like to use, without losing the system's generality. A composer should be able to depart from a perceptually-oriented vocabulary, independently of a synthesis model and then implement it (for example, the concept of maintenance given in Chapter 4, §4.2.2). Further study involving other synthesis techniques is necessary in order to test whether it is either possible to improve our mapping method, or to end up with a more efficient one. The next synthesis technique we plan to employ in our investigation is synthesis by physical modelling (Roads, 1993; Woodhouse, 1992; Keefe, 1992; Smith, 1992; Cook, 1992). This technique matches particularly well the notion of the organisation of the knowledge of sounds based upon a functional modularisation of the mechanism which produces these sounds.

We have confined ourselves to a purely symbolic approach, as we began this research project assuming that the design of an ISSD would essentially involve only the

modelling of certain cognitive problems and not sub-cognitive ones. Now that we have accomplished this, we would like to advance towards hybrid systems, by adding the support of an "analogical", sub-symbolic level to the symbolic one. Neural networks technology is suitable for this task. We propose that a neural network could be able to efficiently identify prominent classificatory features in input samples of (real) sounds and also provide ways of referring to them using user-defined labels. Neural network technology (Forrest et al., 1987) combined with the phase vocoder resynthesis technique (Wishart, 1993; Dobson, 1993) would also constitute an interesting ground to investigate better links between intuitive sound descriptions and synthesis parameters.

Regarding the interfacing problem, we plan to add visually-oriented means of communication to the current command line-oriented one. There is, however, another consideration here: if we deliberately provide menu items (or buttons) for each operation the program can support, then the user interface inversely becomes cluttered and harder to use as the software becomes more sophisticated. Thus, the interfacing problem will not simply be sorted out by providing a variety of menu items. Further research is needed in order to find a balanced solution for this.

Finally, with reference to support for the process of exploration, we plan to provide a mechanism for the automatic generation of *scripts* that retain information related to the exploratory steps, performed during a certain design process (in a similar fashion as in the Elthar program - Chapter 3, §3.3). These scripts would be available, so that they could be recalled when a similar design process is demanded in other situations.

Bibliography

Arcela, A. (1994), A Linguagem SOM-A para Síntese Aditiva, in *Proceedings of the I Simpósio Brasileiro de Computação e Música,* Caxambú, pp. 33-43, SBC, Brazil.

Berio, L. (1966), *Sequenza III*, Compact Disk 426 662 - 2, Aufnahme/Phillips Classics Production (1970).

Barrière, J-B. (1989), Computer music as a cognitive approach: Simulation, timbre, and formal processes, in *Contemporary Music Review*, Vol. 4, pp. 117-130, Harwood Academic Publishers, UK.

Bayle, F. (1993), *musique acousmatique: propositions & positions*, INA/GRM & Buchet/Chastel, France.

Bench-Capon, T. J. M. (1990), *Knowledge Representation: An Approach to Artificial Intelligence*, The A.P.I.C. Series Nr. 32, Academic Press, UK.

Berg, P. (1975), *ASP Report,* Technical Report, Utrecht Institute of Sonology, The Netherlands.

Berg, P. (1980), SSP and Sound Description, in *Computer Music Journal*, Vol. 4, Nr. 1, pp. 25-35, The MIT Press, USA.

Beyls, P (1993), Creativity and computation: Tracing attitudes and motives, in *Proceedings of the 4th International Symposium on Electronic Art,* Minneapolis, pp. 19-28, ISEA, USA.

Bismark, G. von (1971), Timbre of Steady Sounds: Scaling of Sharpness, in *Proceedings of the 7th International Congress on Acoustics*, Budapest, Vol. 3, pp. 637-640, Akadémiai Kiadó, Hungary.

Bismark, G. von (1974a), Timbre of Steady Sounds: A Factorial Investigation of its Verbal Attributes, in *Acustica*, Vol. 30, pp. 146-158, USA.

Bismark, G. von (1974b), Sharpness as an Attribute of the Timbre of Steady Sounds, in *Acustica*, Vol. 30, pp. 159-172, USA.

Bowen, D. L., **Byrd**, L., **Pereira**, F. C. N., **Pereira**, L. M., and **Warren**, D. H. D. (1994), *SICStus Prolog 2.1 #9 User's Manual,* Swedish Institute of Computer Science, Sweden.

Brachman, R. J. and **Levesque**, H. J. (editors) (1985), *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, USA.

Brandão, M.C.P. and **Nascimento**, R. S. R. (1991), A Geometric Concordance Device for Sound Synthesis, in *Proceedings of ICMC Montreal*, pp. 412-415, ICMA, Canada.

Bratko, I. (1990), *Prolog Programming for Artificial Intelligence*, Addison Wesley, USA.

Camurri, A., **Canepa**, C., **Frixione**, M., and **Zaccaria**, R. (1992), HARP: A Framework and a System for Intelligent Composer's Assistance, in *Readings in Computer-Generated Music*, Baggi, D. (editor), IEEE Computer Society Press, pp. 95-115, USA.

Carbonell, J. G. (1990), Introduction: Paradigms for Machine Learning, in *Machine Learning: Paradigms and Methods*, Carbonell, J. G. (editor), The MIT Press/ Elsevier, USA/The Netherlands.

Chion, M. (1983), *Guide des objets sonores*, INA/GRM & Buchet-Chastel, France.

Clarke, M (1988), FOF Synthesis on the Atari ST, in *Composers' Desktop Project software documentation*, available from 11, Kilburn Road, York, UK.

Cogan, R. (1984), *New Images of Musical Sound*, Harvard University Press, USA/UK.

Cook, P. R. (1993), SPASM, a Real-Time Vocal Tract Physical Model Controller, in *Computer Music Journal*, Vol. 17, Nr. 1, pp. 36-41, The MIT Press, USA.

Cosi, P., **De Poli**, G., and **Lauzzana**, G. (1994), Auditory Modelling and Self-Organizing Neural Networks for Timbre Classification, in *Journal of New Music Research*, Vol. 23, pp. 71-78, Swets & Zeitlinger, The Netherlands.

Dietterich, T. and **Michalski**, R. (1981), Inductive Learning of Structural Descriptions, in *Artificial Intelligence*, Vol. 16, Elsevier, The Netherlands.

Dodge, C. and **Jerse**, T (1985), *Computer Music - Synthesis, Composition, and Performance*, Schirmer Books, USA.

Dobson, R. (1993), The Operation of the Phase Vocoder, in *Composers' Desktop Project software documentation*, available from 11, Kilburn Road, York, UK.

Dowling, W. J. and **Harwood**, D. L. (1986), *Music Cognition*, Academic Press, UK/USA.

Edmonds, E. (1993), Culture, Knowledge and Creativity - beyond computable numbers, in *Languages of Design*, Vol. 1, Nr. 3, pp. 253-261, Elsevier, The Netherlands.

Ehresman, D. and **Wessel**, D. (1978), *Perception of Timbral Analogies*, Rapports IRCAM, Centre Georges Pompidou, France.

Eiche, J. F. (1987), *What's a Synthesiser?*, Hal Leonard Books, USA.

Ethington, R. and **Punch**, B. (1994), SeaWave: A System for Musical Timbre Description, in *Computer Music Journal*, Vol. 18, Nr. 1, pp. 30-39, The MIT Press, USA.

Farvis, K. M. and **Macintosh**, A. L. (1988), *An Introduction to UNIX Systems in The University of Edinburgh* (4th edition), University of Edinburgh, UK.

Feiten, B. and **Ungvary**, T. (1990), Sound Data Base Using Spectral Analysis Reduction And An Additive Synthesis Model, in *Proceedings of ICMC Glasgow 1990,* pp. 72-74, ICMA, UK.

Feiten, B. and **Günzel,** S. (1993), A Sound Retrieval Index based on Two Dimensional Similarity Maps, in *Proceedings of the 94th Convention of the Audio Engineering Society*, Berlin, pre-print 3542(F1-10), AES, Germany.

Feiten, B. and **Günzel,** S. (1994), Automatic Indexing of a Sound Database Using Self-Organizing Neural Nets, in *Computer Music Journal*, Vol. 18, Nr. 3, pp. 53-65, The Mits Press, USA.

Flanagan, F. (1984), Voices of Men and Machines, in *Electronic Speech Synthesis*, Bristow, G. (editor), Granada, UK.

Fodor, J. A. and **Pylyshyn**, Z. W. (1988), Connectionism and Cognitive Architecture, in *Connections and Symbols*, Pinker, S. and Meher, J. (editors), The MIT Press, USA.

Forrest, B. M., **Roweth**, D., **Stroud**, N., **Wallace**, D. J., and **Wilson**, G. V. (1987), *Neural Networks Models*, Physics Dept. pre-print 87/419 (ECSP - TR - 11), University of Edinburgh, UK.

Garton, B. (1989), The Elthar Program, in *Perspectives of New Music*, Vol. 27, Nr. 1, pp. 6-41, USA.

Giomi, F. and **Ligabue**, M. (1992), *Analisi Assistita al Calcolatore della Musica Contemporanea*, Rapporto Interno C92-01, CNUCE/CNR, Conservatorio di Musica L. Cherubini, Italy.

Haton, J-P. and **Haton**, M-C. (1989), *L'Intelligence Artificielle*, Que sais-je? series Nr. 2444, Presses Universitaires de France, France.

Hayes-Roth, B. (1983), Using Proofs and Refutations to Learn from Experience, in *Machine Learning*, Vol. 1, pp. 221-240, Tioga, USA.

Helmholtz, H. L. F. (1885), *On the sensations of tone as a physiological basis for the theory of music*, Longmans, Green and Co, UK.

Heslop, B. D. and **Angell**, D. F. (1993), *Mastering Solaris™ 2*, Sybex, USA.

Holtzman, S. (1978), *A description of an automated digital sound synthesis instrument*, DAI Research Report Nr. 59, Dept. of AI, University of Edinburgh, UK.

Howel, P, **Cross,** I., and **West,** R. (editors) (1985), *Musical Structure and Cognition,* Academic Press, UK.

Jaffe, D. (1994), An Overview of Criteria for Evaluating Synthesis and Processing Techniques, in *Proceedings of the I Simpósio Brasileiro de Computação e Música,* Caxambú, pp. 53-61, SBC, Brazil.

Johnston, I., (1989), *Measured Tones: The interplay of physics and music*, IOP Publishing, UK.

Kaegi, W. and **Tempelars,** S. (1978), VOSIM - A new sound synthesis system, in *Journal of Audio Engineering Society,* Nr. 26, pp. 418-425, USA.

Kasabov, N. K. and **Petkov**, S. (1992), Neural Networks and Logic Programming - a Hybrid Model and its Applicability to Building Expert Systems, in *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, pp. 287-288, John Willey & Sons, UK.

Keefe, D. H. (1992), Physical Modelling of Wind Instruments, in *Computer Music Journal*, Vol. 16, Nr. 4, pp. 57-73, The MIT Press, USA.

Klatt, D. H. (1980), Software for a cascade/parallel formant synthesiser, in *Journal of the Acoustic Society of America*, Vol. 67, Nr. 3, pp. 971-995, USA.

Kohonen, T. (1980), Self association and memory, in *Series in Information Science*, Vol. 8, Springer Verlag, Germany/UK.

Laird, E., **Newell**, A., and **Rosenbloom**, A. (1987), SOAR: an Architecture for General Intelligence, in *Artificial Intelligence*, Vol. 33, pp. 1-64, Elsevier, The Netherlands.

Laurent, J-P. (1992), Proposals for a Valid Terminology in KBS Validation, in *Proceedings of the 10th European Conference on Artificial Intelligence,* Vienna, pp. 829-834, John Willey & Sons, UK.

Lerdahl, F. (1987), Timbral Hierarchies, in *Contemporary Music Review*, Vol. 2, pp. 135-160, Harwood Academic Publishers, UK.

Logan, B., **Corne**, D., and **Smithers**, T. (1992), The Edinburgh Designer System: An Architecture for Solving Ill-structured Problems, in *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, pp. 282-286, John Wiley & Sons, UK.

Lohner, H. (1986), The UPIC System: A User's Report, in *Computer Music Journal,* Vol. 10, Nr. 4, pp. 42-49, The MIT Press.

Luger, G. F. and **Stubblefield**, W. A. (1989), *Artificial Intelligence and the Design of Expert Systems*, The Benjamin/Cummings Publishing Company, USA/Canada.

Mâche, F-B. (1991), *Musique, Mythe, Nature ou les dauphins d'Arion*, Méridiens Klincksieck, France.

Marino, G. (1990), The New UPIC System, *in Proceedings of ICMC Glasgow 1990,* pp. 249-252, ICMA, UK.

Marino, G., **Serra**, M-H., and **Raczinski**, J-M. (1993), The UPIC System: Origins and Innovations, in *Perspectives of New Music*, Vol. 31, Nr. 1, pp. 258-269, USA.

Mathews, M. and **Pierce**, J. (1987), The Computer as a Musical Instrument, in *Scientific American*, February/87, pp. 90-97, USA.

McAdams, S. (1987), Music: A science of the mind?, in *Contemporary Music Review*, Vol. 2, pp. 1-61, Harwood Academic Publishers, UK.

McAdams, S., and **Deliège**, I. (editors) (1985), *Contemporary Music Review: Music and the Cognitive Sciences,* Vol. 4, UK.

McClelland, J. and **Rumelhart**, D. (1986), *Parallel Distributed Processing*, The MIT Press, USA.

Miranda, E. R. (1992), T*owards an Acousmatic Singer,* Research Report Nr. 1, Faculty of Music, University of Edinburgh, UK.

Miranda, E. R. (1993), Modelagem do Aparelho Fonador e suas Aplicações na Música, in *Acústica & Vibrações,* Journal of the Acoustic Society of Brazil, Nr. 12, pp. 60-74, Brazil.

Miranda, E. R. (1995), Granular Synthesis of Sounds by means of a Cellular Automaton, in *Leonardo Music Journal*, USA (in press).

Miranda E. R., **Nelson**, P., and **Smaill**, A. (1992), ChaOs, a Model for Granular Synthesis by means of Cellular Automata, in *EPCC Annual Report 91-92 & Project Directory*, pp. 153-156, Edinburgh Parallel Computing Centre, UK.

Morrison, J. and **Waxman**, D. (1991), *Mosaïc 3.0*, Ircam, France.

Newell, A. (1981), The Knowledge Level, in AI Magazine, Vol. 1, Nr. 3, pp. 1-20, AAAI, USA.

Oliveira, M. A. O. (1981), *Kant*, Cadernos da UnB, Editora Universidade de Brasília, Brazil.

Orton, R. (1988), ADSYN DRAW - User Manual, in *Composers' Desktop Project software documentation*, available from 11, Kilburn Road, York, UK.

Palombini, C. (1992), *Pierre Schaeffer's typo-morphology of sonic objects: the method for musical research,* PhD Thesis, Faculty of Music, University of Durham, UK.

Palombini, C. (1993a), Machine Songs V: Pierre Schaeffer - From Research into Noises to Experimental Music, in *Computer Music Journal*, Vol. 17, Nr. 3, pp. 14-19, The MIT Press, USA.

Palombini, C. (1993b), Pierre Schaeffer, 1953: Towards an Experimental Music, in *Music & Letters*, Vol. 74, Nr. 4, Oxford University Press, UK.

Petitot, J. (1989), Perception, cognition and morphological objectivity, in *Contemporary Music Review*, Vol. 4, pp. 171-180, Harwood Academic Publishers, UK.

Piaget, J. (1947), *La psychologie de l'intelligence*, Colin, Paris, France.

Plotkin, G. (1970), A note on inductive generalisation, in *Machine Intelligence*, Meltzer, B. and Michie, D. (editors), Edinburgh University Press, UK.

Puckette, M., **Lippe**, C., and **Waxman**, D. (1992), *ISPW Max Reference Manual,* Preliminary Release 0.17, Ircam, France.

Quinlan, J. R. (1986), Induction of decision trees, in *Machine Learning*, Vol. 1, Nr. 1.

Rademakers, P. and **Pfeifer**, R. (1992), The Role of Knowledge Level Modules in Situated Adaptive Design, in *Proceedings of the 10th European Conference on Artificial Intelligence,* Vienna, pp. 601-602, John Willey & Sons, UK.

Risset, J-C. (1992), Composing Sound with Computers, in *Companion to Contemporary Musical Thought*, Paynter, J. et al. (editors), Vol. 1, pp. 583-621, Routledge, UK.

Roads, C. (1993), Initiation à la Synthèse par Modèles Physiques, in *La Synthèse Sonore*, Les cahiers de l'Ircam Nr. 2, pp. 145-172, Editions Ircam - Centre Georges Pompidou, France.

Roads, C. (1980), Interview with Marvin Minsky, in *Computer Music Journal*, Vol. 4, Nr. 3, pp. 25-39, The MIT Press, USA.

Rodet, X., **Potard**, Y., and **Barrière**, J-B. (1984), The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General, in *Computer Music Journal*, Vol. 3, Nr. 3, pp. 15-31, The MIT Press, USA.

Roozendal, R. (1993), Psychological analysis of musical composition: composition as design, in *Contemporary Music Review*, Vol. 9, Parts 1 & 2, pp. 311-324, Harwood Academic Publishers, UK.

Rosen, K. H., **Rosinski**, R. R., and **Farber**, J. M. (1990), *UNIX System V - Release 4*, Osborne/MacGraw-Hill, USA.

Rossing, T. D. (1990), *The Science of Sound*, Addison Wesley, USA/Canada.

Schaeffer, P. (1966), Traité des objets musicaux, Ed. du Seuil, France.

Scipio, A. D. (1994), Formal Processes of Timbre Composition - Challenging the Dualistic Paradigm of Computer Music (A Study in Composition Theory II), in *ICMC Proceedings 1994*, Århus, pp. 202-208.

Schmidt, B. L. (1987), Natural Language Interface and their application to Music Systems, in *Proceedings of the 5th Audio Engineering Society International Conference*, pp. 198-206, AES, USA.

Schottstaedt, W. (1992), *Common Lisp Music Documentation*, available via Internet ftp from the clm directory on the host machine ccrma-ftp.Stanford.edu.

Schottstaedt, B. (1994), Machine Tongues XVII: CLM: Music V Meets Common Lisp, in *Computer Music Journal*, Vol. 18, Nr. 2, pp. 30-37, The MIT Press.

Seifert, U. (1991), The schema concept - a critical review of its development and current use in Cognitive Science and research on Music Perception, in *Proceedings of the IX Colloquio di Informatica Musicale*, Associazione di Informatica Musicale Italiana, pp. 116-131, Italy.

Slawson, W. (1985), *Sound Color*, University of California Press, USA/UK.

Slawson, W. (1987), Sound-color Dynamics, in *Perspectives of New Music*, Vol. 25, Nrs. 1 & 2, pp. 156-179, USA.

Sloboda, J. A. (1985), *The musical mind: the cognitive psychology of music*, Oxford University Press, USA.

Smaill, A., **Wiggins,** G. A., and **Miranda**, E. R. (1994), Music Representation - between the Musician and the Computer, in *Music Education: An Artificial Intelligence Approach*, Smith, M. et al. (editors.), Workshops in Computing Series, Springer-Verlag, pp. 108-119, UK.

Smith, J. O. (1992), Physical Modelling using Digital Waveguides, in *Computer Music journal*, Vol. 16, Nr. 4, pp. 78-47, The MIT Press, USA.

Smithers, T., **Conkie**, A., **Doheny**, J., **Logan**, B., and **Millington**, K. (1989), *Design as Intelligent Behaviour: An AI in Design Research Programme*, Unpublished draft, Dept. of AI, University of Edinburgh, UK.

Spender, N. (1980), Psychology of music (I-III), in *The New Grove's Dictionary of Music and Musicians*, Sadie, S. (editor), Vol. 15, pp. 388-427, Macmillan Publishers, UK.

Sundberg, J. (1991), Synthesising Singing, in *Representation of Musical Signals*, De Poli et al. (editors), The MIT Press, USA.

Suppes, P. and **Crangle**, C. (1990), Robots that learn: A test of Intelligence, in *Revue Internationale de Philosophie*, Nr. 152, pp. 5-23, Université Libre de Bruxelles, Belgium.

Tang, P. C. L. (1984), On the Similarities between Scientific Discoveries and Musical Creativity: A Philosophical Analysis, in *Leonardo*, Vol. 17, Nr. 4, pp. 261-268, Pergamon Press, UK.

Terhardt, B. (1974), On the Perception of Periodic Sound Fluctuation (Roughness), in *Acustica*, Vol. 30, pp. 201-213, USA.

Todd, P. M. and **Loy,** D. G. (editors) (1991), *Music and Connectionism*, The MIT Press, USA.

Vercoe, B. (1991), *Csound Manual,* available via Internet ftp from the music directory on the host machine media-lab.mit.edu.

Wessel, D. L. (1979), *Timbre space as a musical control structure*, Rapports IRCAM, Centre Georges Pompidou, France.

Winkhuyzen, R. E. (1992), On the Non-Existence of Knowledge Level Model, in *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, pp. 620-622, John Willey & Sons, UK.

Winston, P. (1984), *Artificial Intelligence*, (2nd edition), Addison-Wesley.

Winston, P. (1985), Learning Structural Descriptions from Examples, in *Readings in Knowledge Representation*, Brachman, R. and Levesque, H. (editors), Morgan Kaufmann Publishers, USA.

Wishart, T. (1985), *On Sonic Art,* Imagineering Press, UK.

Wishart, T. (1993), The Phase Vocoder: Introduction and Reference Manual, in *Composers' Desktop Project software documentation*, available from 11, Kilburn Road, York, UK.

Wisnick, J. M. (1989), *O Som e o Sentido*, Cia. das Letras, Brazil.

Woodhouse, J. (1992), Physical Modelling of Bowed Strings, in *Computer Music Journal,* Vol. 16, Nr. 4, pp. 43-56, The MIT Press, USA.

Xenakis, I. (1963), Musiques Formelles, in *La Revue Musicalle,* double issue Nos. 253 & 254, Editions Richard-Masse, France.

Xenakis, I. (1971), *Formalized Music: Thought and Mathematics in Music Composition*, Indiana University Press, USA.

Xenakis, I. (1992), *Formalized Music: Thought and Mathematics in Music*, Pendragon Press (revised edition), USA.

Appendix I

The signal processing of the case study instrument

In this appendix we present the signal processing of the case study instrument. This instrument produces human-voice like sounds using subtractive synthesis. As we mentioned in Chapter 4, §4.2., we believe that at this initial stage of our investigation, it is important to work with a model of sound production whose functioning resembles the functioning of its acoustic counterpart, in this case, the functioning of the human voice mechanism. The subtractive synthesis technique suits our purposes here. We already mentioned in Chapter 5, §5.2 that the way in which the signal processing architecture is designed, plays an important role in the definition of the vocabulary for sound description. As we shall see in the following paragraphs, the subtractive synthesis technique allows us to design a signal processing architecture whose modules perform specific tasks associated with parts of the human voice mechanism. In this case, the vocabulary for sound description would probably be biased towards this mechanism.

1 The signal processing block diagram

Our example study instrument supports both parallel and serial filter connection (Chapter 4, §4.2.2.). Its block diagram is shown in Figure I.1. Each part will be separately introduced in the following sections. Their Csound (Vercoe, 1991) programming code is found in Appendix II. It is recommended to consult Appendix II as the reader goes through the following paragraphs.

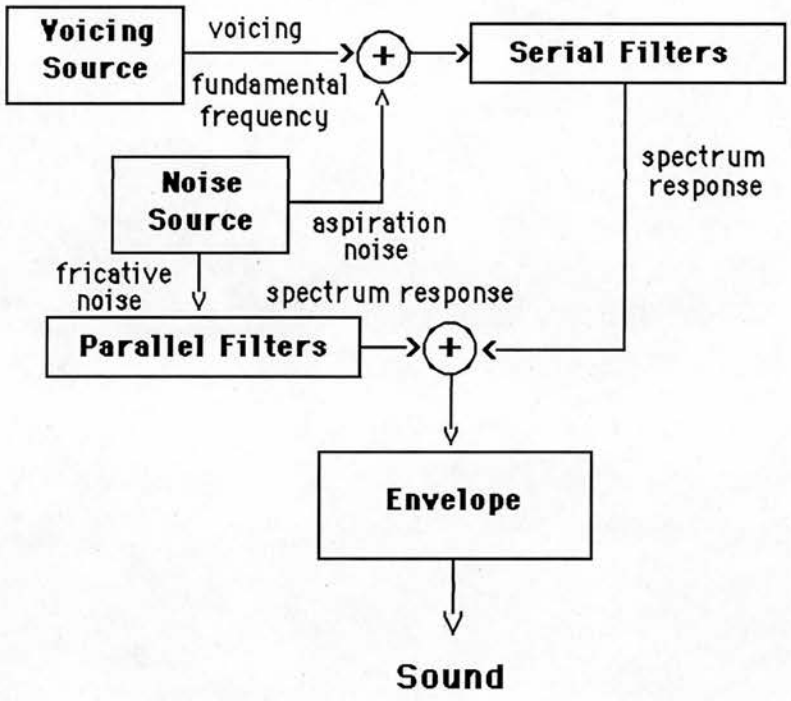
1.1 The sound sources

When singing or speaking, an airstream is forced upwards through the trachea from the lungs. At its upper end, the trachea enters the larynx, which in turn opens into the pharynx. At the base of the larynx, the vocal folds are folded inwards from each side, leaving a variable tension and a slit-like separation, both controlled by muscles in the

larynx. In normal breathing, the folds are held apart to permit the free flow of air. In singing or speaking, the folds are brought close together and tensed. The forcing of the airstream through the vocal folds in this state sets them into vibration. As a result, the airflow is modulated at the vibration frequency of the vocal folds (Sundberg, 1991).

Despite the fact that the motion of the vocal folds is not a simple but a non-uniform vibration, the pitch of the produced sound is determined by this motion.

Figure I.1: The synthesiser's block diagram.



There are two kinds of sound sources in our synthesiser: the *voicing source*, which produces a quasi-periodic vibration, and the *noise source,* which produces turbulence.

The voicing source generates a pulse stream intended to simulate the non-uniform (or quasi-periodic) vibration of the vocal folds, whereas the noise source is intended to simulate an airflow past a constriction or past a relatively wide separation of the vocal folds.

1.1.1 The voicing source

Jitter and *vibrato* are very important for voicing sound quality. Jitter and (non-uniform) vibrato add a degree of non-uniformity to the fundamental frequency of the sound source.

Jitter is defined as the difference in fundamental frequency from one period of the sound to the next. It normally varies at random between -6% and +6% of the fundamental frequency (Sundberg, 1991). In our implementation this value is obtained by adding the results from three random numbers generators whose values are produced by interpolating periodic random values.

Vibrato is defined as the frequency modulation of the fundamental frequency. Vibrato is interesting from a timbral point of view. In humans it is important for the recognition of the identity of the singer. Typically, there are two distinct parameters for vibrato control: *the amplitude* (or *width*) of the vibrato, that is, the amplitude of the modulating frequency, and *the frequency* (or *rate*) of the vibrato, that is the frequency of the modulating frequency. Our implementation selects at random a value between 3% and 6% of the fundamental frequency for vibrato width (after Klatt (1980)).

The heart of the voicing source is a Csound unit called *buzz*, which is a pulse generator. It produces a periodic wave form at a specific frequency with a great deal of energy in the harmonics. A pulse waveform has significant amplitude only during relatively brief time intervals, called *pulse width* (Figure I.2). When a pulse waveform is periodically repeated, the resulting signal has a rich spectrum. The spectrum of a pulse waveform has both odd and even harmonics, being greater for waves with a shorter duty cycle. The output of the voicing source provides the raw material from which the filtering system will shape the required sound.

The human singer controls the spectrum envelope by adjusting the muscles of the larynx, controlling the spacing and tension of the vocal folds. In loud singing the vocal folds close completely for a significant fraction of the vibration cycle. The resulting airflow has a waveform like the one shown in Figure I.3(a). If the folds are too slack, they do not close properly at any point in the vibration cycle. Here the resulting waveform loses its "spiky" appearance and starts to resemble a sinusoidal curve, like that shown in Figure I.3(b). This results in the breathy sound quality often

found among untrained singers. "Trained singer" here means one with the ability to maintain the vocal folds pressed tightly together.

Figure I.2: Generalised waveform of a pulse. The pulse width determines the richness of the spectrum. The narrower the pulse width, the richer the spectrum.

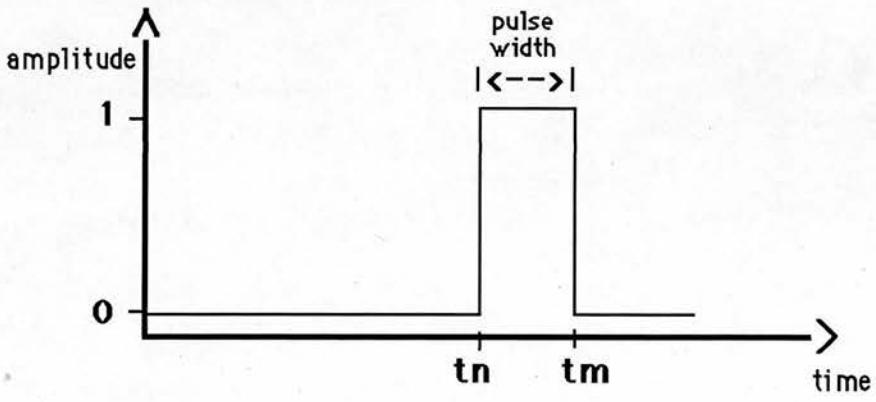
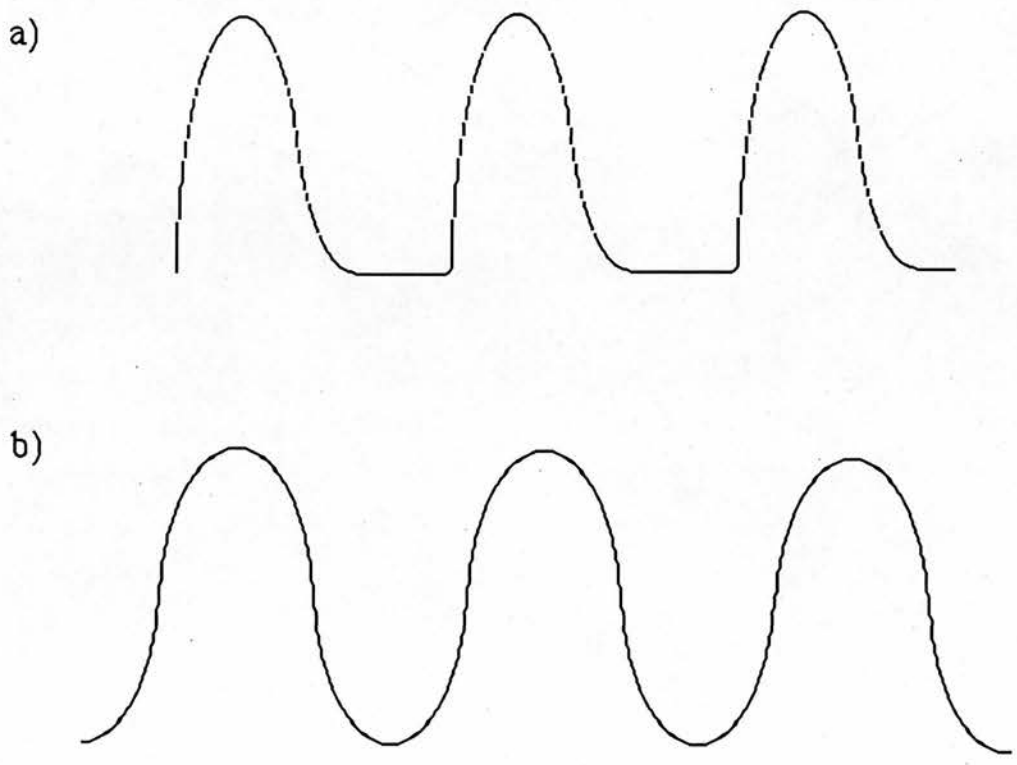


Figure I.3: (a) normal voicing, (b) quasi-sinusoidal voicing.



The output of the voicing source is the sum of two kinds of voicing generators: *normal voicing* and *quasi-sinusoidal voicing*. Two trains of pulses are sent through a low-pass filter (LPF) in order to produce a smoother waveform. The filtered pulses then have a spectrum that falls off at approximately 12 dB/oct above half value of the bandwidth value. The appropriate shape for quasi-sinusoidal voicing is obtained using another LPF. One-half of the bandwidth value determines the cut-off frequency beyond which the harmonics are attenuated. Some degree of quasi-sinusoidal voicing may be added to the normal voicing source, in combination with aspiration noise, in order to produce the aforementioned breathy sound quality.

1.1.2 The noise source

Noise in the human voice corresponds to the generation of turbulence by the airflow past a constriction and/or past a relatively wide separation of the vocal folds. The resulting noise is called *aspiration* if the constriction is located at the level of the vocal folds (larynx). If the constriction is located above the larynx, the resulting noise is called *fricative*. If it is produced by a relatively wide separation of the vocal folds then the resulting noise is called *bypass*.

The synthesiser produces noise using a random number generator, a modulator, and an amplitude controller. It generates the three kinds of noises: fricative, aspiration, and bypass. The fricative noise is sent to the parallel filters, whereas the aspiration noise is added to the voicing source to feed the cascade filters. The bypass noise is the same as the fricative noise, but it is not filtered (Klatt, 1980).

A single random generator is used for both fricative and aspiration noise. Note that in this case the maximum amplitude value of the random generator will be 1, but this does not mean 1 dB. It actually generates values between 0 and 1, which will be used to scale a certain dB value later on. We also implemented two switches: one for fricative noise and one for aspiration noise. They determine whether the noise is to be modulated (amplitude modulation) or not. The fricative noise, if not modulated, will be scaled later according to the desired response of the parallel filters. The degree of modulation is fixed at 50%. That is to say that if the fricative is modulated, half of the amplitude is given by the modulator oscillator and half by the modulated noise. The modulation envelope is a square wave and the modulation rate is the same as the fundamental frequency of the sound being synthesised. For the aspiration noise, the

mechanism is almost the same. The difference is that the aspiration noise is scaled according to a given amplitude value. Modulated noise results in a kind of pitched noise, also known as *pink noise*.

1.2 The filtering system

On its journey through the vocal tract, the sound is transformed. Components which are close to one of the resonance frequencies of the tract are transmitted with high amplitude, while those which lie far from a resonance frequency are suppressed. Much of the art of the singer lies in shaping the vocal tract in such a way that the crude source output is transformed into a desired sound. The vocal tract can be thought of as a tube from the vocal folds to the lips plus a side branch leading to the nose, with a cross-section area which changes considerably.

The length and shape of a particular human vocal tract determine the resonance in the spectrum of the voice signal. The length of the tube is typically about 17 cm (Sundberg, 1991), which can be slightly varied by rising or lowering the larynx and shaping the lips. The cross-sectional area of the tube is determined by the placement of the lips, jaw, tongue and velum. For the most part, however, the resonance in the vocal tract is tuned by changing its cross-sectional area at one or more points. A variety of sounds may be obtained by adjusting the shape of the vocal tract during phonation.

1.2.1 The serial filters

Five band-pass filters (BPF) are appropriate for simulating a vocal tract with length of about 17 cm. A typical female's vocal tract is 15 to 20% shorter than that of a typical male. Klatt (1980) suggests that four filters only are enough to simulate a female's tract. Thus we must allow the removal of the fifth filter from the cascade branch. Each filter (Csound unit called *reson*) introduces a peak in the magnitude spectra determined by the passband centre frequency and by the formant bandwidth. The passband centre frequencies and bandwidths of the lowest three formants vary substantially with changes in articulation, whereas the fourth and fifth formants do not vary as much.

When singing vowels, the connecting passage between the throat and the nasal cavity is closed by raising the back and the soft palette. Opening this passage while singing, produces a sound usually described as *singing through the nose*. The resonance of the nasal cavity is of great importance for the production of some consonants.

The side branch leading to the nasal cavity may be simulated by introducing an anti-resonator Band-Reject Filter (BRF) and a "nasal" sixth BPF (Klatt, 1980). The effect of a BRF and a "nasal" BPF is twofold. They introduce notches in the spectrum and modify the amplitude of the formants. In other words, nasalization introduces additional "hills and valleys" (Chapter 4, §4.2.2.) in the spectrum envelope of a sound.

A Csound *balance* unit is placed at the output of the cascade filter connection in order to amplify the overall midband gain. By taking a reference value, this unit modifies the amplitude of the output signal of the last filter of the cascade so that the average power of the latter is equal to the average power of the former.

1.2.2 The parallel filters

In the parallel connection we do not implement an equivalent filter for the first formant. Although it is possible to simulate a cascade configuration using a parallel configuration, our synthesiser does not consider this possibility. Here the parallel configuration should be mainly used to supply the cascade connection with certain sound qualities and not to produce a sound on its own.

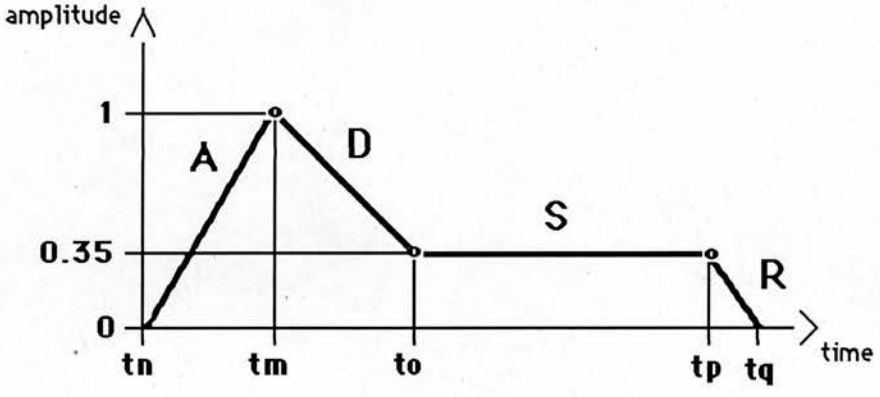
As explained above, only the fricative noise is sent through parallel filters. The amplitude of the fricative noise is individually scaled before filtering. In addition, we implemented a bypass path with its own individual amplitude value. The bypass is used to introduce a harsh flat noise, named *bypass noise*.

Passband centre frequencies and bandwidths for the parallel connection are the same as for cascade connection.

1.3 The envelope

The way a sound event varies in amplitude is called *envelope*. An envelope is normally represented as a simple graph of amplitude (vertical) versus time (horizontal). Figure I.4 shows the implemented envelope. Here a sound event has four distinct sections: **A**ttack, **D**ecay, **S**ustain and **R**elease.

Figure I.4: The implemented envelope scheme.



Appendix II

The Csound code for the case study instrument

We mentioned in Appendix I that the signal processing architecture of the instrument plays an important role in the specification of the knowledge of the ISSD. In Appendix II, we complement this discussion by presenting the Csound code (that is, the orchestra file) of the case study instrument illustrated in Figure I.1, in Appendix I. Here we can observe which p-fields belong to which block of the instrument and foresee how we can associate these p-fields with the slots of the Abstract Sound Schema (ASS), introduced in Chapter 6.

We present the code for each module separately and present its synthesis parameters.

1 The modules

The Voicing Source

Jitter

(1) krnd1 randi .02, .05
(2) krnd2 randi .02, .111
(3) krnd3 randi .02, 1.219
(4) ksum=krnd1+krnd2+krnd3
(5) kfnd=kf0+(kf0*ksum)

Vibrato

(1) kaw rand .03
(2) kaw=abs(kaw)+.03
(3) kvib oscil (kfnd*kaw), kvr, 1
(4) kfund=kvib+kfnd

The Voicing Source

```
(1) knh=int(8000/kf0)
(2) apulsev   buzz kav, kfund, knh, 1
(3) apulseqv buzz kqs, kfund, knh, 1
(4) alpfgv  reson  apulsev,  0, kbgp*1.414, 1
(5) alpfgqv  reson  apulseqv,  0, kbgp*1.414, 1
(6) alpfqs  reson  alpfgqv,  0, kbgs*1.414, 1
(7) anv  balance alpfgv, apulsev
(8) aqsv balance alpfqs, apulseqv
(9) avoicing=anv+aqsv
```

The Noise Source

```
(1) arnd rand 1
(2) if p21=0 goto nomodfric
(3) amodf oscil .5, kfund, 2
(4) afric=(.5*arnd)+amodf
(5) goto aspir
(6) nomodfric:
(7)       afric=arnd
(8) aspir:
(9)       kah=kah/2
(10)      if p22=0 goto  nomodaspir
(11)      amodh oscil kah, kfund, 2
(12)      aspir=(kah*arnd)+amodh
(13)      goto transfunc
(14)      nomodaspir:
(15)            aspir=kah*arnd*2
(16) transfunc:
```

Filtering System

The Serial Filters

```
(1) abrfzc=avoicing+aspir
(2) if p17=0 goto allpole
(3) abpfnc reson avoicing+aspir, kfnp, kbnp, 1
(4) abrfzc areson abpfnc, kfnz, kbnz, 1
(5) allpole:
(6)     abpf1c reson abrfzc, kf1, kbw1, 1
(7)     abpf2c reson abpf1c, kf2, kbw2, 1
(8)     abpf3c reson abpf2c, kf3, kbw3, 1
(9)     abpf4c reson abpf3c, kf4, kbw4, 1
(10)    if p18=1 goto male
(11)    abpf5c=abpf4c
(12)    goto balancing
(13)    male:
(14)        abpf5c reson abpf4c, kf5, kbw5, 1
(15)    balancing:
(16)        acascout balance abpf5c, avoicing+aspir
```

The Parallel Filters

```
(1) abpf2p reson afric*ka2, kf2, kbw2, 1
(2) abpf3p reson afric*ka3, kf3, kbw3, 1
(3) abpf4p reson afric*ka4, kf4, kbw4, 1
(4) abpf5p reson afric*ka5, kf5, kbw5, 1
(5) abypass=afric*kab
(6) aparout=abpf2p+abpf3p+abpf4p+abpf5p+abypass
```

The Envelope

```
(1) kenv linseg katck, ka*kdur, kdecay, kd*kdur, ksust, ks*kdur,
        krels, kr*kdur, kend
(2) aout=(acascout+aparout)*kenv
```

2 The synthesis parameters

Each synthesis parameter below corresponds to a slot of the ASS (Chapter 6, §6.2.1) of the instrument of our ISSD example (Chapter 7, §7.1.1 and §7.1.2).

Voicing Source:

start = Start time of the sound event in seconds
duration = Duration of the sound event in seconds
f0 = Fundamental frequency of the sound in Hz
bgp = LPF's cutoff frequency in Hz
bgs = LPF's cutoff frequency in Hz
av = Amplitude of normal voicing in dB
avs = Amplitude of quasi-sinusoidal voicing in dB
vibr = Vibrato rate in Hz

Noise Source:

ah = Amplitude of aspiration signal

Cascade (or Serial) Filters:

fnp = Nasal BPF's passband centre frequency in Hz
bnp = Nasal BPF's bandwidth in Hz
fnz = Anti-resonator's passband centre frequency in Hz
bnz = Anti-resonator's bandwidth in Hz
f1 = 1st formant's passband centre frequency in Hz
bw1 = 1st formant's bandwidth in Hz
f2 = 2nd formant's passband centre frequency in Hz
bw2 = 2nd formant's bandwidth in Hz
f3 = 3rd formant's passband centre frequency in Hz
bw3 = 3rd formant's bandwidth in Hz
f4 = 4th formant's passband centre frequency in Hz
bw4 = 4th formant's bandwidth in Hz
f5 = 5th formant's passband centre frequency in Hz
bw5 = 5th formant's bandwidth in Hz

Parallel Filters:

a2 = 2nd formant's amplitude in dB
a3 = 3rd formant's amplitude in dB
a4 = 4th formant's amplitude in dB
a5 = 5th formant's amplitude in dB
ab = Bypass amplitude in dB

Envelope:

a_begin = Attack begin value between 0 and 1
a = Attack duration in seconds
d_begin = Decay begin value between 0 and 1
d = Decay duration in seconds
s_begin = Sustain begin value between 0 and 1
s = Sustain duration in seconds
r_begin = Release begin value between 0 and 1
r = Release duration in seconds
r_end = Release end value between 0 and 1

Switches:

sw1 = (ON) means adds side branch "nasal resonators"
sw2 = (ON) means sex is male or (OFF) for female
sw3 = (ON) means adds vibrato and jitters
sw4 = (ON) means amplitude modulation of fricative noise
sw5 = (ON) means amplitude modulation of aspiration noise

Appendix III

Studying the role of each parameter using a synthesis example

In Appendix III we study the role of each synthesis parameter (as listed in Appendix II, §3) using an example. We would like to emphasise here the great amount of knowledge and expertise required to produce a single sound in such a synthesiser. The computer here is aimed at helping the composer to manage this knowledge.

The control parameters which usually vary for synthesising an isolated vowel are: **av**, **f0**, **f1**, **f2**, **f3**, **f4**, **f5**, **bw1**, **bw2**, **bw3**, **bw4**, and **bw5**. The fourth and fifth formants are not essential for high intelligibility, thus they can remain constant for all vowel-like sounds.

In order to create a natural breathy vowel, one must activate the amplitude of aspiration, **ah** and quasi-sinusoidal voicing, **avs**. The amplitude of the voicing source, **av**, is usually set to a value not less than **60 dB**. Phonation frequency, **f0** is set to any desired frequency, but it should not be higher than **f1**. To achieve a good result **f0** is usually set to approximately one-half of **f1**.

Let us take the vowel /a/ (as in the word "bat" in English) as an example. The first formant centre frequency for the vowel /a/ is **622.25 Hz** and the second is **1028 Hz** (see Appendix IV for passband centre frequency and bandwidth values for other vowels). Thus **f0** is set equal to **311.12 Hz**.

The voicing source's output is sent through a Low-Pass Filter (LPF) to produce a smooth waveform which resembles a typical glottal waveform (Klatt, 1980). The parameter **bgp** must be set as a double value of the desired cutoff frequency. The filtered impulses thus have a spectrum that falls off smoothly at **12 dB** per octave above **bgp / 2**. This cutoff frequency is not usually set lower than an octave above the phonation frequency. Good results are obtained by setting this cutoff value to the same as the passband centre frequency of the first formant. So let us set **bgp** equal to **1.3 kHz**.

The parameter **avs**, determines the amount of smoothed voicing to be added to the normal voicing in order to produce a breathy sound quality. A reasonable result is obtained by setting this value to **6 dB** below **av** (**avs = av - 6**). Thus if we set to **av** equal **86 dB**, then **avs** is set to **80 dB**. The wave shape for the quasi-sinusoidal voicing is obtained by double low-pass filtering the signal in order to have a fall off at **24 dB** per octave after the cutoff frequency (double low-pass filter here means two second order filters connected in series). The output of the above LPF is fed to a second LPF whose cutoff frequency, **BGS**, is usually set to approximately an octave above **bgp**. In our example, **bgs**, is set equal to **2093.4 Hz**. The parameter **ah** is also an amplitude value but it controls the amount of aspiration noise to be added to a normal voicing for breathy sound quality. For a moderate noisy breath, **ah** is set to **40 dB**.

The parameter **vibr** specifies the vibrato rate. Vibrato is essentially a frequency modulation of **f0**. Here the vibrato rate is set equal **5.2 Hz**. If **vibr** is set above the audio range (approximately **18 Hz**) the aural effect produced will not be a vibrato but a distortion - which may be useful to produce a wide range of interesting effects.

Nasal murmurs and vowel nasalization are approximated by the insertion of additional BPF (pole or resonator) and BRF (zero or anti-resonator) into the cascade filters. The parameters that are used to generate a nasal murmur, include the nasal pole and zero bandwidths **bnp** and **bnz**, and frequencies **fnp** and **fnz**. Remember that the spectrum envelope of a speech like sound looks like a pattern of hills and valleys. Technically, each hill corresponds to the resonance of a pole transfer function, whereas each valley corresponds to a zero transfer function. The BPF and BRF are primarily used to approximate vowel nasalization, by splitting the first formant **f1** into a pole-zero-pole complex. A nasalized vowel is generated by fixing **fnz** to, let us say, **270 Hz**, increasing **f1** to about **100 Hz** and setting **fnp** to be the average of the new **f1** value and that of **fnz**. Both **bnp** and **bnz** may be set as the same as **bw1**. Our vowel is not, however, to be nasalized. There are two ways of avoiding nasalization: by switching off **sw1** (this immediately passes the signal through the all-pole cascade) or by setting **fnz = fnp** and **bnz = bnp**.

The fricative's scaling amplitudes for the parallel filters were all set to **0**. This means that we are not dealing with fricative noise or parallel filters at the moment.

The most fundamental way in which a sound varies during its course, is the variation in its overall amplitude, that is, the dynamic envelope. This can have far more drastic effects on the nature of a sound than one might think. The envelope setting for our example is as follows:

a_begin	**= 0**	(i.e., begins with total attenuation)
a	**= 0.05**	(i.e., attack lasts for 5% of **duration**)
d_begin	**= 1**	(i.e., goes up to full amplitude)
d	**= 0.01**	(i.e., decay lasts for 1% of **duration**)
s_begin	**= 0.9**	(i.e., suffers 10% of attenuation)
s	**= 0.79**	(i.e., sustain lasts for 79% of **duration**)
r_begin	**= 0.85**	(i.e., ends sustain with 15% of attenuation)
r	**= 0.15**	(i.e., release lasts 15% of **duration**)
r_end	**= 0**	(i.e., finishes with total attenuation)

There are five switches (named **sw1** to **sw5**) used to connect or separate certain modules to the main architecture. A switch only has two states: it can be either **1** (for **ON**) or **0** (for **OFF**). The parameter **sw1** adds the side branch nasal resonators (6th BPF and BRF) at the beginning of the cascade. As we do not want nasalization in this example, we set **sw1 = 0**. We set **sw2 = 1** to switch on the 5th filter of the cascade because we are synthesising a male voice. The parameter **sw3 = 1** adds jitters and vibrato to **f0**. The settings **sw4 = 1** and **sw5 = 1** indicate that the fricative and aspiration noise, respectively, are to be modulated by the fundamental frequency.

See Appendix IV for centre formant frequencies and bandwidths values for the vowels /e/, /i/, /o/, and /u/. The other parameters for these vowels can be set as for the vowel /a/ (explained above).

Appendix IV

Passband centre frequencies and bandwidths for vowels

In Appendix IV, we provide the centre frequencies and the bandwidth values for five vowels (these values are the result of our own experiments with the synthesiser described in Appendix I):

(a) vowel /a/, as in the word "bat" in English.
(b) vowel /e/, as in the word "blend" in English.
(c) vowel /i/, as in the word "if" in English.
(d) vowel /o/, as in the word "roll" in English.
(e) vowel /u/, as in the word "full" in English.

Table 1: Passband centre frequencies

	1	2	3	4	5
/a/	650	1028	2650	2900	3250
/e/	400	1700	2600	3200	3580
/i/	290	1870	2800	3250	3540
/o/	400	800	2600	2800	3000
/u/	350	600	2700	2900	3300

Table 2: Bandwidths

	1	2	3	4	5
/a/	80	90	120	130	140
/e/	70	80	100	120	120
/i/	40	90	100	120	120
/o/	40	80	100	120	120
/u/	40	60	100	120	120

Appendix V

An example operation

In this appendix we work through an example operation. We present this in the form of a tutorial. The reader should refer to the user's manual, given in Appendix VII for more information on the commands exemplified below.

We begin by explaining how to load and run the system. We illustrate a training session, followed by examples of how to consult the knowledge of the system. Next we illustrate how to design sounds. Here we will imagine three sounds and show how we could design them. Finally, we illustrate the functioning of the learning by introspection mechanism and end the example operation by showing how to halt the system.

1 Loading the system

At the Music Faculty of Edinburgh University, ARTIST runs on a Sun Sparc 10 Station computer under the Solaris™ operational system (Heslop and Angell, 1993). We suggest operating the program on a single shell (of Solaris' Open Windows) and to enable its scrolling. This provides a means to examine what has been done throughout a working session, simply by scrolling the window up or down.

To run ARTIST, firstly one calls Prolog and then loads the program. In order to load the program one types '[artist]', as shown below:

```
|?-[artist].
```

Before displaying the main menu, ARTIST will display various messages saying that the program is being loaded. The main menu is shown below.

ARTIST displays:

```
============================================================
University of Edinburgh  -  Faculty of Music & Dept. of AI
============================================================
Artificial Intelligence-based Synthesis Tool - A R T I S T

V 1.1 -  Experimental Prototype  - May 94
============================================================

============================================================
ARTIST commands for sound design:  --  Main Menu
------------------------------------------------------------
Type [p] to play a known sound.
Type [ds] to derive a sound with slots.
Type [da] to derive a sound with attributes.
Type [dsa] to derive a sound with slots and attributes.
Type [cs] to create a novel sound with slots.
Type [ca] to create a novel sound with attributes.
Type [csa] to create a novel sound with slots and attributes.
Type [fgt] to forget a sound.

Type [i] for information service.
Type [ml] for machine learning engine.
Type [r] to remember last session.

Type [la] to listen again the last synthesised sound.
Type [halt] to halt.
============================================================
Enter your choice:
```

2 Training the system

ARTIST must be trained at the start of a new working session (alternatively the user can activate the [**r**] command to remember the last session, if any - see the user's manual in Appendix VII). In order to train the system, we have to call the machine learning engine menu by means of the [**ml**] command.

The user inputs:

Enter your choice: **ml.**

Then the machine learning engine menu appears as shown below.

ARTIST displays:

```
============================================================
ARTIST learning engine:
------------------------------------------------------------------------------
Type [iscd] to display the induced shortest concept description of a sound.
Type [t] to learn from a training set.
Type [it] to learn by introspection.
Type [tset] to display the knowledge for introspection.
Type [skp] to go back to main menu.
------------------------------------------------------------------------------
Enter your request:
```

The next step is to select the [**t**] command. This command enables us to teach a training set to the system. Here ARTIST will ask the name of the training set (the training set for this example is shown in Appendix VI).

The user inputs:

Enter your request: **t.**

Enter information:

Name of the training set: **training_set.**

At this point, ARTIST displays messages connected with the training process, followed by other messages about the status of the learning algorithm, as shown below.

ARTIST displays:

Learning status:

Inducting rules about vowel_a.
Inducting rules about vowel_e.
Inducting rules about vowel_i.
Inducting rules about vowel_o.
Inducting rules about vowel_u.

2.1 Consulting a rule

In order to consult the induced rule for a certain sound, say **vowel_a**, we have to select the [**iscd**] command, as illustrated below.

The user inputs:

Enter your request: **iscd.**

Enter information:

Name of the sound: **vowel_a.**

ARTIST displays:

The ISCD rule for sound vowel_a is as follows:
acuteness = medium.

Let us now go back to the main menu and explore other features of the system. In order to recall the main menu, we type the **[skp]** command.

3 Exploring the information of the knowledge base

Here we illustrate how to explore the knowledge in the system. We call the information service menu by activating the **[i]** command (of the main menu). The information service menu is then displayed as shown below.

ARTIST displays:

```
============================================================
ARTIST information service:
------------------------------------------------------------
Type [ks] to list known sounds.
Type [ats] to list the attribute values of a sound.
Type [sls] to list the slot values of a sound.
Type [nsls] to list the numerical slot values of a sound.
Type [ka] to list known attributes.
Type [atv] to list known attribute values.
Type [slv] to list the slot value(s) of an attribute value.
Type [nslv] to list the numerical slot value(s) of an attribute value.
Type [th] to display the meaning of a slot rule.
Type [skp] to go back to main menu.
============================================================
Enter your request:
```

3.1 Consulting the names of known sounds

In order to consult the names of known sounds, we activate the [**ks**] command. In our case, ARTIST should display the information as shown below.

ARTIST displays:

The list of known sounds:

vowel_a.
vowel_e.
vowel_i.
vowel_o.
vowel_u.

3.2 Consulting the available attributes for sound description

In order to check the available attributes for sound description, we select the [**ka**] command. The following information is then displayed.

ARTIST displays:

The list of known attributes:

acuteness.
envelope.
formant_1.
formant_2.
fundamental_frequency.
openness.
radiation.
roughness.
vibrato.

3.3 Consulting which adjectives can be used to describe a certain attribute

Suppose that we wish to know which attribute values (or adjectives) the system understands for the attribute **register**. Here we use the [**atv**] command.

The user inputs:

Enter your request: **atv.**

Enter attribute: **register.**

ARTIST displays:

The list of known attribute values for register:

high.
low.
medium.
medium_frequency.
high(exceptionally).
high(extremely).
high(in_theory).
high(moderately).
high(very).
high(exceptionally(in_theory)).
high(extremely(in_theory)).
high(moderately(in_theory)).
high(very(in_theory)).
low(extremely).
low(in_theory).
low(very).
low(extremely(in_theory)).
low(very(in_theory)).
medium(high).
medium(in_theory).
medium(low).
medium(typical).

The [**nslv**] command must be activated, if the user wishes to check the "meaning" of an adjective in terms of synthesis parameter values (that is, slot values). Let us examine how ARTIST interprets the attribute *theoretically very low register.*

The user inputs:

Enter your request: **nslv.**

Enter attribute=value: **register=low(very(in_theory)).**

ARTIST displays:

The list of slot values for the attribute value register=low(very(in_theory)):

f(0)= rule(f(0),low(very(in_theory))).

Here ARTIST indicates that there is only one slot, namely **f(0)**, tied to the attribute **register**. It also indicates that the value of this slot is given by a rule. In order to check how this value is calculated, we use the **[th]** command.

The user inputs:

Enter your request: **th.**

Enter slot=value: **f0=low(very(in_theory)).**

ARTIST displays:

Rule: very low f0 theoretically is 3 octaves below f1.

4 Designing and playing sounds

Suppose that we wish to design three sounds: two sound of three seconds each (the default duration value of this particular implementation) and one of five seconds.

We assume that the user is aware of the available vocabulary for sound description at this point. A reasonable intuition of the capabilities of the system is also helpful here. That is, the user should at least know that this system produces human-voice like sounds using subtractive synthesis.

The first sound is to be a kind of distorted vowel in a low register. We can achieve distortion here by adding some noise and fast vibrato to a vowel. For the second sound, we vaguely imagine a sound with medium acuteness. Any sound with medium acuteness should be satisfactory here. For the third sound, we would like to design a breathy, 440 Hz, open vowel-like sound, with very slow vibrato, lasting five seconds.

Let us now design these sounds. But firstly we illustrate how to play a known sound. Then we suggest how to design the above sounds.

4.1 Playing a known sound

In order to play a known sound, that is, a sound present in the knowledge base (and in the sound files directory, that is, assuming however that it was previously synthesised), we simply type the [**p**] command of the main menu and then input the name of the sound.

The user inputs:

Enter your choice: **p.**

Enter information:

Name of the known sound: **vowel_a.**

Having heard how the sound **vowel_a** sounds like, let us now attempt to derive our first sound example from it.

4.2 Deriving a sound from another sound

Here we will attempt to design the "distorted vowel" sound by changing the attributes of the sound **vowel_a**. In this case, we use the [**da**] command. We will name this sound as **distorted_vowel**.

The user inputs:

Enter your choice: **da.**

Enter information. Type [skip] to skip.

Name of the new sound: **distorted_vowel.**
Name of the original sound (type [ks] for help): **vowel_a.**

Attribute (type [ka] for help): **register.**
Value (type [atv] for help): **low.**

Attribute (type [ka] for help): **vibrato.**
Value (type [atv] for help): **distortion(typical).**

Attribute (type [ka] for help): **roughness.**
Value (type [atv] for help): **noisy.**

Attribute (type [ka] for help): **skip.**

At this point, ARTIST should activate the Csound compiler and produce the new sound by changing: the register, the vibrato and the roughness of the sound **vowel_a**. Note that the sound *vowel_a* has not in fact suffered any transformation itself. The system actually synthesises a new sound using the synthesis parameters of *vowel_a*, but changes the information provided in the requirement.

Let us now try to create the second sound and examine how the system uses its induced rules.

4.3 An example of ARTIST's use of its rules

Now we will design the second sound, by creating if from scratch (using the **[ca]** command), rather than deriving it from another known sound. This example will demonstrate how ARTIST uses its induced rules.

The user inputs:

Name of the new sound: **medium_acuteness.**

Attribute (type [ka] for help): **acuteness.**
Value (type [atv] for help): **medium.**

Attribute (type [ka] for help): **skip.**

ARTIST displays:

The sound vowel_a is an example of:

acuteness=medium.

ARTIST will play vowel_a instead of medium_acuteness.

In this case, ARTIST found a rule stating that the sound **vowel_a** may fulfil our requirement. Thus, instead of producing a new sound, it plays the sound **vowel_a**. Let us assume that we are satisfied with this result and study how we can design the third sound of our operation example.

4.4 Creating a novel sound by inputting slots and attributes

Now we will produce the "breathy 440 Hz open sound". Here we also opted to create the sound from scratch. In this case we need to explicitly specify the fundamental frequency and duration values. We will also specify slot values for the first formant centre frequency (**f1**) and bandwidth (**bw1**). Let us name this sound as **open_sound**.

The user inputs:

Enter your choice: **csa.**

Name of the new sound: **open_sound.**

Slot: **duration.**
Value: **5.**

Slot: **f1.**
Value: **660.**

Slot: **bw1.**
Value: **30.**

Slot: **f1.**
Value: **440.**

Slot: **skip.**

Attribute (type [ka] for help): **roughness.**
Value (type [atv] for help): **breathy.**

Attribute (type [ka] for help): **vibrato.**
Value (type [atv] for help): **slow(very).**

Attribute: **skip.**

ARTIST displays:

ARTIST knows no rule about a sound whose attribute value(s) is(are):

roughness=breathy.
vibrato=slow(very).

ARTIST used default sound values.

In this case, ARTIST "knows" no rule which fulfils the requirement (in terms of attribute values). Hence a new sound has been created. Missing slot values were

completed using default values (inherited from the default sound - see the user's manual, §1.2.1., in Appendix VII).

5 Learning by introspection

We are now in a position to study how the task of learning by introspection works. This is triggered by the [**it**] command of the machine learning engine menu. Note that the system will deduce that the values we input for the first formant centre frequency (**f1**) and bandwidth (**bw1**) (when we designed the sound **open_sound**) do not match any known attribute value for **openness** and **formant_1**. Thus, the system will create new attribute values for them and will ask us to name these new values (for example, an adjective in English). In case we do not wish to name a new deduced attribute value, ARTIST will still consider it to be a new attribute value, but will record it with an indication that no name has been given.

The user inputs:

Enter your request: **it.**

ARTIST displays:

Wait please! ARTIST is thinking ...

ARTIST deduced an unknown attribute value for "openness" whose slot values are:

bw1=30.
f1=660.

Would you like to give a name for this novel attribute value?

Type [y] for yes.
Type [n] for no.

The user inputs:

Enter your choice: **y.**

Name of the new attribute value: **very_open.**

ARTIST displays:

ARTIST deduced an unknown attribute value for "formant_1" whose slot values are:

bw1=30.
f1=660.

Would you like to give a name for this novel attribute value?

Type [y] for yes.
Type [n] for no.

The user inputs:

Enter your choice: **y.**

Name of the new attribute value: **high_and_narrow.**

Then ARTIST displays messages concerned with the training procedure followed by the status of the introspection process:

Learning status:

Inducting rules about distorted_vowel.
Inducting rules about open_sound.
Inducting rules about vowel_a.
Inducting rules about vowel_e.
Inducting rules about vowel_i.
Inducting rules about vowel_o.
Inducting rules about vowel_u.

6 Finishing a working session

To finish a working session, we type the [**halt**] command of the main menu. Here ARTIST will ask if we wish make an introspection. As we have just done this, there is no need to make an introspection again.

The user inputs:

Enter your choice: **halt.**

ARTIST displays:

Attention! ARTIST will halt.

Would you like to make an introspection before?

Type [y] for yes.
Type [n] for no.

The user inputs:

Enter your choice: n.

ARTIST displays:

Bye.

aida%

Appendix VI

The example operation training set

```
training_set([

sound_event(vowel_a,
            [       acuteness = medium,
                    envelope = default,
                    formant_1 = vowel(a),
                    formant_2 = vowel(a),
                    fundamental_frequency = medium(in_theory),
                    openness = high,
                    radiation = normal,
                    register = medium(in_theory),
                    roughness = default,
                    vibrato = default
            ]),

sound_event(vowel_e,
            [       acuteness = high,
                    envelope = default,
                    formant_1 = vowel(e),
                    formant_2 = vowel(e),
                    fundamental_frequency = medium(in_theory),
                    openness = high(very),
                    radiation = normal,
                    register = medium(in_theory),
                    roughness = default,
                    vibrato = default
            ]),

sound_event(vowel_i,
            [       acuteness = high(very),
                    aspiration_noise = medium,
                    envelope = default,
                    formant_1 = vowel(i),
                    formant_2 = vowel(i),
                    fundamental_frequency = medium(in_theory),
                    openness = low,
                    radiation = normal,
                    register = medium(in_theory),
                    roughness = default,
                    vibrato = default
            ]),
```

```
sound_event(vowel_o,
            [       acuteness = low,
                    envelope = default,
                    formant_1 = vowel(o),
                    formant_2 = vowel(o),
                    fundamental_frequency = medium(in_theory),
                    openness = medium,
                    radiation = normal,
                    register = medium(in_theory),
                    roughness = default,
                    vibrato = default
            ]),

sound_event(vowel_u,
            [       acuteness = low(very),
                    envelope = default,
                    formant_1 = vowel(u),
                    formant_2 = vowel(u),
                    fundamental_frequency = medium(in_theory),
                    openness = low(very),
                    radiation = normal,
                    register = medium(in_theory),
                    roughness = default,
                    vibrato = default
            ])      ]).
```

Appendix VII

ARTIST 1.2 User's Manual

In Appendix VII we provide the user's manual of ARTIST prototype version 1.1. This is a thesis appendix, therefore the basics of ARTIST is not introduced as it would be in a working manual.

The manual is divided in two parts; Part 1 and Part 2. Part 1 focuses upon the implementation issues of the user-specified modules (Chapter 7, §7.1). Part 2 focuses upon how to operate the system.

1 Part 1: The instrument design level

1.1 Implementing the user-specified modules

ARTIST currently runs on a Sparc 10 Sun Computer, under the Solaris™ operational system (Heslop and Angell, 1993) and SICStus Prolog (Bowen et al., 1994).

All files of the system (including the user-specified modules) should be placed together in one single directory (for example, /klang/artist/). A folder called *sounds* should be created in this directory (for example, /klang/artist/sounds/). ARTIST will save sound files there.

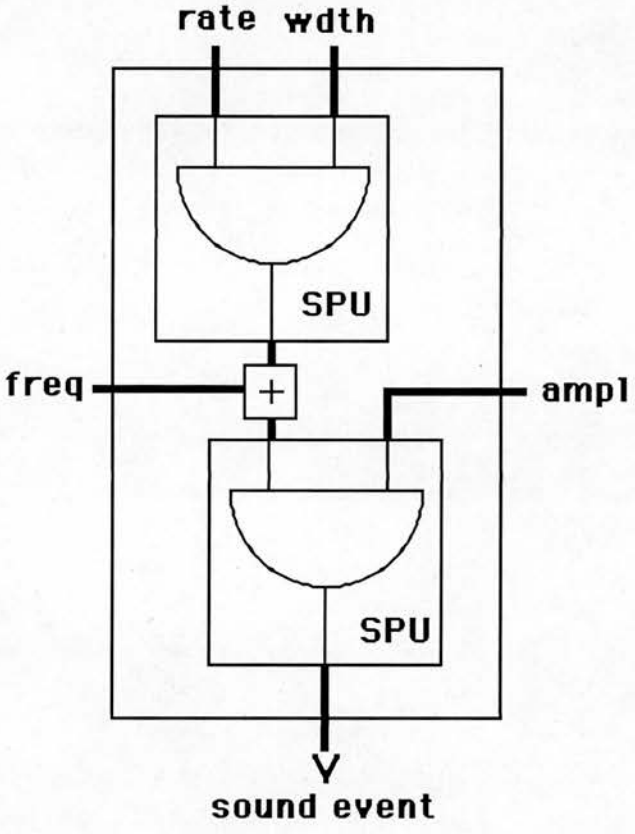
The user-specified modules are implemented in Prolog (Bratko, 1990; Bowen et al. 1994). Each module is saved in a different file and should be named according to the instructions following.

We examine the syntax of each module through the implementation of simple examples.

1.1.1 The instrument and the schema modules

Suppose that we wish to implement an ISSD (Intelligent Assistant for Sound Design) whose signal processing has the following architecture (Figure V.1):

Figure V.1: The ISSD example.



This synthesiser has two Signal Processing Units (SPU): a *frequency modulator*, which is used as a vibrato generator and an *oscillator*. In the current version of ARTIST, the instrument is implemented in Csound (available via internet ftp from the music directory on the host media-lab.mit.edu). ARTIST translates its output into a Csound score file. The Csound orchestra file for this instrument is as follows:

```
sr = 44100
kr = 100
ksmps = 441
nchnls = 2
instr1
a1 oscil p4, p5, 1,
a2 oscil p6, p7+a1, 1
outs a2, a2
endin
```

The Csound p-fields **p1**, **p2** and **p3** (which correspond to the instrument number, start time and duration time (both in seconds), respectively) are automatically set by ARTIST: **p1** and **p2** value by default 1 and 0, respectively. The user does not have access to these settings. However, **p3** is automatically tied to a slot called **duration**. The slot **duration** need not be specified by the user in the *schema* module; but the user may specify a vocabulary to describe it in the *knowledge base* (to be dealt with in §1.2.) and indeed refer to it in a requirement when designing a sound. If the user makes no reference to the slot **duration** whatsoever, then ARTIST sets it to a default value: 3 seconds.

The orchestra file should be saved as *art.orc*. All the tables and other Csound GEN units should be specified in a different file - observing the Csound score file syntax - and be saved as *art.sco*. When producing a sound, ARTIST opens this file, adds the p-fields values and runs the Csound compiler.

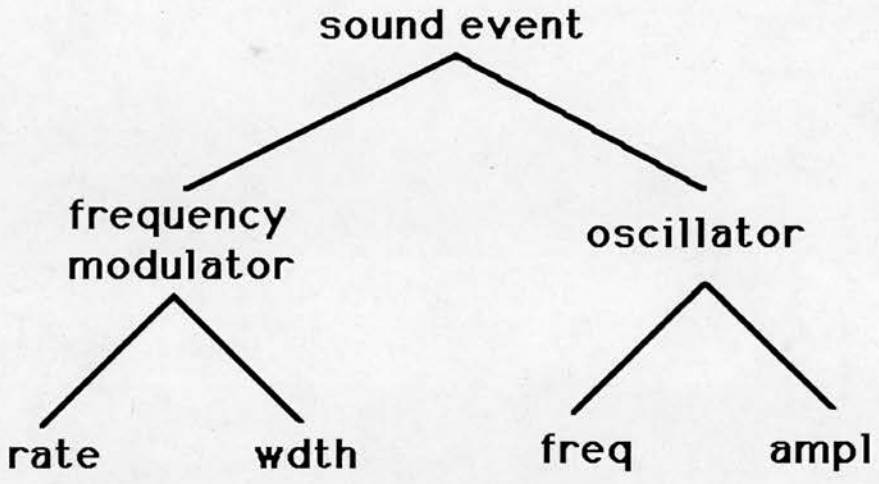
The ASS schema for this synthesiser may be defined as illustrated in Figure V.2.

We say that this synthesiser has four slots: **rate**, **wdth**, **freq** and **ampl**. We can associate three sound attributes to the SPUs: **vibrato** (to the frequency modulator branch) and **fundamental frequency** and **loudness** (to the oscillator branch).

The implementation of the ASS, uses the clause *has_component* with two arguments. It specifies the offspring relation between slots and attributes. The second argument is the offspring of the first argument. See example below:

```
has_component( sound_event, vibrato).
has_component( sound_event, oscillator ).
has_component( oscillator, fundamental_frequency ).
has_component( oscillator, loudness ).
has_component( vibrato, rate ).
has_component( vibrato, wdth ).
has_component( fundamental_frequency, freq ).
has_component( loudness, ampl ).
```

Figure V.2: The ASS schema.



Observe that we have tied the attribute **vibrato** to the node **frequency modulator** and the attributes **fundamental frequency** and **loudness** to the slots **freq** and **ampl**, respectively.

It is essential that the order of appearance of the slots in the ASS (considering the specification of the *has_component* clauses from top down) follows the same order of the p-fields in the Csound orchestra file (for instance, **p4 = rate**, **p5 = wdth**, **p6 = freq** and **p7 = ampl**). The other nodes of the ASS do not necessarily need to follow any order though.

The *schema* module must be saved as a file named *schema*.

1.2 The knowledge base module

The *knowledge base* module holds clusters of slot values which are used by ARTIST to instantiate a sound or an attribute. To implement these clusters, the user chooses the clause *slot* with two arguments. The first argument represents the name of the cluster, whereas the second argument represents the slot values. See example below:

```
slot( sound_event( sound(x), [ rate, 5.2 ] ).
slot( sound_event( sound(x), [ wdth, default ] ).
slot( sound_event( sound(x), [ freq, medium ] ).
slot( sound_event( sound(x), [ ampl, 78 ] ).
%
slot( sound_event( sound(y) ), [ a_kind_of, sound(x) ] ).
slot( sound_event( sound(y) ), [ freq, high] ).
%
slot( attribute( fundamental_frequency = high ), [ freq, high ] ).
%
slot( attribute( loudness = piano ), [ ampl, low ] ).
%
slot( attribute( vibrato = default ), [rate, 5.2 ] ).
slot( attribute( vibrato = default) , [wdth, default ] ).
```

There are two kinds of slot clusters in the knowledge base: one which assembles a *sound event* and another which assembles an *attribute*. The term "sound event" here, refers to the instantiation of the root of the ASS schema, that is, the instrument, whereas "attribute" means an instantiation of a node of the schema, that is, a component of the instrument at any level. The difference between them is represented by using either the clause *sound_event* or *attribute*, respectively. Values for the slots can be either a number or a word. The meaning of a word should be specified in the dictionary module (and eventually with a link to the theory module) (see §1.1.3 and §1.1.4, below).

Note that the collection of slots for the sound event **sound(y)** is different from **sound(x)**; it looks incomplete. It contains, however, a different sort of information. The new information is a link called *a_kind_of*. This is a link which associates one collection of slots with another. In this case, **sound(y)** inherits missing information

from **sound(x)**. That is, the only difference between **sound(y)** and **sound(x)** is that the former has a higher frequency.

It is worth considering that clusters of slots for a node (that is, for an attribute) must not be incomplete. In this version of the system, there is no inheritance mechanism for attribute values. ARTIST is not yet able to identify the attribute if some slot(s) is (are) missing.

We remind the reader that, there is in fact no need to specify slots for sound events in the knowledge base. Instead, the user could specify the sound events in terms of their attribute values only. ARTIST is able to work out the meaning of the attribute values of a sound event in terms of slot values. The attributes, however, should have been previously specified. This is dealt later, in §1.2, when we study the specification of a training set for ARTIST.

The *knowledge base* module must be saved as a file named *slots*.

1.2.2.1 The default sound

ARTIST requires the specification of a default sound. Even if the user never wants to synthesise it, this sound should be present in the knowledge base. The default sound should be named as **default** (for example, **sound_event(default)**). The slots of this sound should be specified with default slot values. These default slot values will be recalled when ARTIST eventually needs to fill the missing slots of those sounds whose inheritance does not work (for example, for new created sounds from an incomplete list of slot and/or attribute values). It is imperative that a complete cluster of slots (that is, with no missing slots) should be used for the **default** sound.

1.2.3 The dictionary module

In this module the user specifies the meaning of each word of the vocabulary for slot values. Each word of the vocabulary may mean either a synthesis parameter value, or a pointer to a rule which contains a formula for calculating it. These rules must be specified in the *theory* module (see §1.1.4 below).

For each entry of the dictionary, the user uses the clause *dict*. By means of this clause the user specifies a list the words of the vocabulary for a slot, followed by their respective meaning. The clause *dict* has two arguments. The first argument (the clause *slot* with one argument) stands for the name of the slot whose values are being defined. The second argument is a list of *value* clauses, whose first argument is the word and the second is its "meaning" (that is, a number value or a pointer to a rule). The pointer to a rule is represented by means of the clause *rule*, whose first argument is the name of the current slot and the second is the word. See examples for the slots **rate** and **freq** as follows:

```
dict( slot( rate ), [   value( slow,          3.4 ) ,
                        value( normal,        5.2 ),
                        value( fast,          8.3 ) ] ).

dict( slot( freq ), [   value( low,           220 ),
                        value( medium,        rule( freq, medium ) ),
                        value( high,          rule( freq, high ) ) ] ).
```

The *dictionary* module must be saved as a file named *dictionary*.

1.2.4 The theory module

In this module, the user specifies a theory for the instrument. A theory is a collection of statements which impose constraints on the functioning of the instrument. There are two kinds of statement: a formula for calculating a slot value (that is, a synthesis parameter) based on other slot values and a formula for calculating a slot value by means of the random choice within a particular interval (here the user may write Prolog procedures).

In order to specify a formula based on other slot values, one writes the *instrument_theory* procedure, with two arguments. The first argument is the reference to the rule (the same as the second argument of the clause *value*, of the list of values of the clause *dict* of the dictionary, discussed in §1.1.3). In order to ascertain the slot values involved in the computation of a rule (in case this value is referred to by a word), ARTIST provides an inbuilt procedure, called *get_value,* which requires three arguments. The first argument is the name of the slot, the second is the value of this

slot and the third is a variable (which holds the value to be used in the computation of the rule). Example:

```
instrument_theory( rule( freq, medium ), Fval ):-
        get_value( freq, low, V ),
        Fval is V * 2.
```

In the example above, **Fval** is the double value of **V** (low frequency). The procedure *get_value* computes the value of **V**.

In order to specify a formula to calculate a value using the random choice within a certain interval, one writes the procedure *random_theory* with four arguments. The first argument is the reference to the rule (the same as the second argument of the clause *value*, of the list of values of the clause *dict* of the dictionary, §1.1.3). The second argument is the lower limit of the choice interval. The third argument is the upper limit of the choice interval. Finally, the fourth argument is a variable which holds the result of the random choice. In the body of this procedure, the user then calls the SICStus Prolog *random* procedure, whose three arguments are the same as the last three arguments of the *random_theory* procedure. See example below:

```
random_theory( rule( freq, high), 680, 1080, Fval ):-
        random( 680, 1080, Fval ).
```

The first of the above examples states that a **freq = medium** means the value of **freq = low** multiplied by two. The second example states that **freq = high** is any frequency value between **680 Hz** and **1080 Hz**. The two kinds of procedures may of course be combined. The example below illustrates the use of the *random* procedure within an *instrument_theory* procedure. Here, **freq = unpredictably_low** is any value between **11 Hz** and **220 Hz** (considering that **freq = low** values **220 Hz**). Example:

```
instrument_theory( rule( freq = unpredictably_low ), Fval ):-
        get_value( freq, low, V ),
        random( 11, 220, R ),
        Fval is V / R.
```

This module must be saved as a file named *theory*.

1.3 Making a training set

The training set is a collection of *sound_event* clauses with two arguments: the name of the sound and a complete list of attribute values. By a "complete list" we mean that all the attributes for sound description defined in the *schema* and in the *knowledge base* modules must be specified for each sound. We have three attributes in the case of the example given above, namely: **vibrato**, **fundamental frequency** and **loudness**. All of them should be present in each *sound_event* entry of the training set. See below for an example of a *sound_event* entry:

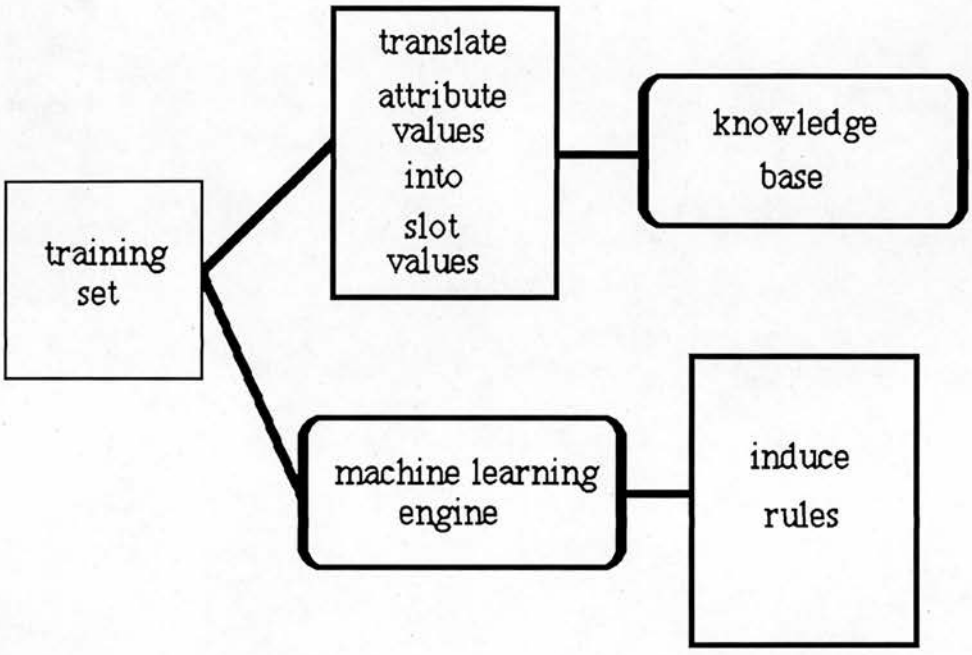
```
sound_event( sound(x), [    vibrato = default,
                            fundamental_frequency = medium,
                            loudness = medium ] ).
```

All *sound_event* clauses of the training set should be placed in a list which is the argument of a clause called *training_set* (see Appendix VI for an example training set). The *training_set* clause is then saved in a file. The user may give any name to this file. Example:

```
training_set(
        [sound_event( sound(x),
                      [vibrato = default,
                      fundamental_frequency = medium,
                      loudness = medium ] ),
        ...,
        ...]     ).
```

ARTIST performs two tasks when the user loads a training set (Figure V.3): it gives the set to the machine learning engine and it translates the set in terms of slot values and records this information in the knowledge base (and also saves a *slot* file - see §1.1.2).

Figure V.3: ARTIST performs two tasks when a training set is loaded.



2 Part 2: The sound design level

2.1 Operating the system

To operate the program, the user firstly calls Prolog and then loads *Artist.pl.* Then the main menu of the system is displayed (see Appendix V, for an example operation).

The current version of ARTIST operates in a dialogue-like manner. That is, after selecting a command of the menu, ARTIST asks the user the information it needs, such as the names of sounds, slot and attribute values and so on. After each "answer" it is necessary to type a full stop (' . ') followed by the 'return' key. Example (main menu):

The user types:	**p + . +** [return key]
ARTIST asks:	*Name of the known sound:*
The user types:	**sound(x) + . +** [return key]

There are three menus of commands: the *main menu*, the *information service menu*, and the *machine learning* menu. Before explaining the commands of each menu, we suggest the following hints:

(**a**) the user may cancel the whole operation of a command at any stage, by typing '**control + C**'. When the Prolog prompt appears, the user types '**engine.**' in order to call back the main menu.

(**b**) when designing a sound, the user types the command '**skip**', in order to terminate the operation or jump to another set of input data (for example, from slot to attribute values). Example:

The user types:	**csa** + **.** + [return key]	
ARTIST asks:	*Enter information. Type [skip] to skip:*	
	Name of the new sound:	
The user types:	**test_sound** + **.** + [return key]	
ARTIST asks:	*Slot:*	
The user types:	**freq** + **.** + [return key]	
ARTIST asks:	*Value:*	
The user types:	**110** + **.** + [return key]	
ARTIST asks:	*Slot:*	
The user types:	**skip** + **.** + [return key]	% jumps to the input of attributes
ARTIST asks:	*Attribute (type [ka] for help):*	
The user types:	**vibrato** + **.** + [return key]	
ARTIST asks:	*Value (type [atv] for help):*	
The user types:	**fast** + **.** + [return key]	
ARTIST asks:	*Attribute (type [ka] for help):*	
The user types:	**skip** + **.** + [return key]	% ends the '**csa**' command

In the above example, ARTIST should produce a 110 Hz sound, with fast vibrato, named as **test_sound**.

2.1.1 The main menu

a) [**p**] *to play a known sound*: plays a sound which was previously synthesised and is present in the knowledge base and in the sound files directory.

b) [**ds**] *derive a sound with slots*: derives a new sound from a known sound but replaces the slot values of the known sound by the new values provided by the user. The new sound is produced and is kept in both the knowledge base and the sound files directory. The user can type the [**ks**] command before inputting the answer, in order to consult the list of the known sounds when the system asks to input the name of the "origin sound".

c) [**da**] *derive a sound with attributes*: derives a new sound from a known sound, but replaces the attribute values of the known sound by the new values provided by the user. The new sound is produced and kept in both the knowledge base and the sound files directory. The user can type the [**ks**] command before inputting the answer, in order to consult the list of the known sounds when the system asks to input the name of the "origin sound" In order to consult the list of known attributes when the system asks to input the name of an attribute, the user can type the [**ka**] command. In the same manner, the user can type the [**atv**] command to consult which values (or adjective) the system knows for a certain attribute.

d) [**dsa**] *derive a sound with slots and attributes*: derives a new sound from a known sound by changing the slot and attributes values of the known sound. It merges the previous two commands ([**ds**] and [**da**]) into a single one. The new sound is kept both in the knowledge base and in the sound files directory. The user can type the [**ks**] command before inputting the answer, in order to consult the list of the known sounds when the system asks to input the name of the "origin sound". In order to consult the list of known attributes when the system asks to input the name of an attribute, the user can type the [**ka**] command. In the same manner, the user can type the [**atv**] command to consult which values (or adjective) the system knows for a certain attribute.

e) [**cs**] *create a new sound with slots*: creates a new sound from a set of given slot values. Note that the probability to successfully create a sound which approximates the user's expectation might be proportional (but not necessarily) to the amount of slot values provided. Missing information will be set to default values (of the **default** sound discussed above in §1.1.2.1). The new sound is kept in both the knowledge base and the sound files directory.

f) **[ca]** *create a new sound with attributes*: creates a new sound from a given set of attribute values. Here ARTIST will consult induced rules in order to check if it knows any sound which might satisfy the user's requirement. If so, ARTIST will play the known sound. Otherwise, ARTIST will attempt to create the new sound based solely upon the given information. Note that the probability to successfully create a sound that approximates the user's expectations might be proportional (but not necessarily) to the amount of attribute values provided. In the latter case, missing information will be set to default values (of the **default** sound discussed above in §1.1.2.1). The new sound is kept both in the knowledge base and in the sound files directory. The user can type the **[ka]** command, in order to consult the list of known attributes when the system asks to input the name of an attribute. In the same manner, the user can type the **[atv]** command to consult which values (or adjective) the system knows for a certain attribute.

g) **[csa]** *create a new sound with slots and attributes*: creates a new sound from the given sets of slot values and attribute values. It merges the two previous commands ([**cs**] and[**ca**]) into a single one. In this case ARTIST also consults induced rules in order to check if it knows any sound which might satisfy the user's requirement (though only in terms of attribute values). If so, ARTIST will play the known sound instead of creating a new sound. Otherwise, ARTIST will attempt to create it from scratch, that is, from the user's input information. In the latter case, missing information will not be set by inheritance but set to default values (of the **default** sound discussed above in §1.1.2.1). Again, the probability to successfully create a sound that approximates the user's expectations might be proportional (but not necessarily) to the amount of attribute values provided. The user can type the **[ka]** command, in order to consult the list of known attributes when the system asks to input the name of an attribute. In the same manner, the user can type the **[atv]** command to consult which values (or adjective) the system knows for a particular attribute.

h) **[fgt]** *to forget a sound*: ARTIST removes a sound from both the knowledge base and the sound files directory.

i) **[i]** *for information service*: activates the information service menu. This module contains various commands for consulting the knowledge of the system. (See the information service menu below in §2.1.2).

j) [**ml**] *for machine learning*: activates the machine learning menu. This menu contains various commands concerned with the machine learning engine of ARTIST. (See the machine learning menu below in §2.1.3).

k) [**r**] *to remember the last session*: retrieves all information generated in the previous working session, such as new induced rules, new sounds, and new attribute values. Unless this command is activated in the beginning of a working session, ARTIST always starts up with only the initial knowledge previously provided in the user specified modules.

l) [**la**] *to listen again the last synthesised sound*: automatically plays the last synthesised sound. The user does not need to input the sound name here.

m) [**halt**] *halt*: finishes a working session. All newly generated information is saved in a file, which can be loaded (or "remembered") in a future session by using the command [**r**] (introduced above).

2.1.2 The information service menu

a) [**ks**] *to list known sounds*: displays the names of all sounds currently in the knowledge base (and consequently in the sound files directory).

b) [**ats**] *to list the attribute values of a sound*: displays the complete list of attribute values of a sound. "Complete list" here, means the complete set of attributes which ARTIST currently knows for sound description (see command [**ka**] below). This operation might take a few moments to be accomplished in case the system needs to deduce attributes values which were not eventually specified during the design of the sound (for example, a missing attribute in a requirement or the existence of a slot value which does not match with any of the known attribute values of the dictionary). If new attribute values are deduced, then the user is asked to name them.

c) [**sls**] *to list the slot values of a sound:* displays the complete list of slot values of a sound, that is, all synthesis parameter values. These values are displayed as words (each word stands for a number value or points to a rule to calculate it) whose "meanings" are specified in the dictionary module.

d) [**nsls**] *to list the numerical slot values of a sound*: also displays the complete list of slot values of a sound, that is, all synthesis parameter values. Here, however, these values are displayed as number values (the actual synthesis parameter values) or as rules to calculate them (these rules are specified in the *theory* module).

e) [**ka**] *to list known attributes*: displays the list of attributes ARTIST knows for sound description.

f) [**atv**] *to list known attribute values*: displays the values (that is, adjectives) ARTIST knows for a certain attribute.

g) [**slv**] *to list the slot value(s) of an attribute value*: displays the slot values of an attribute value. These values are displayed as words (each stands for a number value or points to a rule to calculate it) specified in the dictionary module.

h) [**nslv**] *to list the numerical slot value(s) of an attribute value*: displays the slot values of an attribute value in terms of either the actual synthesis parameter value, or a pointer to a rule for calculation.

i) [**th**] *to display the meaning of a slot rule*: tells us how a rule of the theory module calculates a slot value.

j) [**skp**] *to go back to the main menu*: calls the main menu.

2.1.3 The learning engine menu

a) [**iscd**] *to display the induced shortest concept description of a sound*: displays the Induction of the Shortest Concept Description (ISCD) rule of a sound. (The Induction of Decision Trees (IDT) algorithm (Chapter 6, §6.5.1.2) is not yet fully working in this version of the program)

b) [**t**] *to learn from a training set*: loads a training set and activates the machine learning engine. Sounds given in the training set are also added to the knowledge base (but not in the sound files directory, since they have not yet been synthesised).

c) **[it]** *to learn from introspection*: activates ARTIST's machine learning engine, to learn from introspection in its knowledge base. Here, the system consults the knowledge base in order to build a training set. This command is aimed at updating the induced rules. That is, it induces rules for sounds which were created during the current working session and updates its current rules, considering those new sounds and sounds that were eventually "forgotten" (see command **[fgt]** of the main menu). As for the information service menu command **[ats]**, the **[it]** command may take a few moments to be accomplished. The building process of the training set involves the deduction of attributes, which were not eventually specified for new sounds. Here, the user is also asked to give names for new deduced attribute values.

d) **[tset]** *to display the knowledge for introspection*: displays the training set which would be used for introspection at a given moment. In this case ARTIST will build the training set, as explained above, but will not activate the machine learning engine.

e) **[skp]** *to go back to the main menu*: calls the main menu.

Appendix VIII

Pre-doctoral publications

1992:

From Symbols to Sound: AI-based Investigation of Sound Synthesis, Research Abstract, 10th European conference on Artificial Intelligence, Workshop Notes W12, Austria.

From symbols to sound: AI-based investigations of sound synthesis [Thesis proposal], DAI Discussion Paper No. 117, Edinburgh University, UK.

ChaOs: a Model for Granular Synthesis by means of Cellular Automata, EPCC Annual Report 91-92, Edinburgh University, UK.

1993:

From Symbols to Sound: AI-based Investigation of Sound Synthesis, DAI Research Paper No. 640, Edinburgh University, UK.

Modelagem do Aparelho Fonador para Síntese Digital e suas Aplicações na Música, Journal of the Acoustic Brazilian Society, Brazil.

A Framework for the Evaluation of Music Representation Systems (co-author), Computer Music Journal, Vol. 17, No. 3, The MIT Press, USA.

Music Representation - between the Musician and the Computer (co-author), DAI Research Paper No. 658, Edinburgh University, UK.

A Symbolic Approach for the Design of Intelligent Musical Instruments (co-author), X Reunión Nacional de Inteligencia Artificial, Mexico.

A Knowledge-based approach for the design of Intelligent Musical Synthesisers (co-author), X Simpósio Brasileiro de Inteligência Artificial, Brazil.

1994:

Music Representation - between the Musician and the Computer (co-author), Workshop in Computing Series, Springer Verlag, UK.

ARTIST: an AI-based tool for the Design of Intelligent Assistants to Sound Synthesis, I Simpósio Brasileiro de Computação e Música, Brazil.

möbiUP strip (co-author), Unknown Public, issue 04: Musical Machinery, UK.

From Symbols to Sound: AI-based Investigation of Sound Synthesis, Contemporary Music Review, Vol. 10, Part 2, Harwood Academic Publishers, UK.

The role of Artificial Intelligence in Computer-aided Sound Composition, Journal of Electroacoustic Music, Vol. 8, Sonic Arts Network, UK (in press).

1995 (in press):

An Artificial Intelligence Approach to Sound Design, Computer Music Journal, The MIT Press, USA.

Granular Synthesis of Sounds by means of a Cellular Automaton, Leonardo, The MIT Press, USA.

Appendix IX

A sample of a published paper

An Artificial Intelligence Approach to Sound Design*

Modern computer technology enables the production of a virtually limitless variety of sounds by providing substantial access to the parameter settings of synthesis algorithms. However, the production of sounds by means of a synthesis algorithm is still accomplished in a very old-fashioned way: by inputting streams of numerical values for each single desired sound. Furthermore, these numerical values are usually worked out manually. The imagination of the composer in this case easily becomes vulnerable to time consuming, non-musical tasks.

We think that the power of the computer could also (a) provide the composer with better ways for expressing his or her requests to a synthesis algorithm and (b) provide appropriate aid for the exploration of sonic ideas. We believe that this situation can be significantly improved by providing computer sound synthesis technology with AI techniques.

The purpose of our research work (Miranda, Smaill, and Nelson, 1993a; 1993b) (Miranda, 1993a; 1994a; 1994b) is to devise an intelligent sound synthesis system that (a) allows the design of sounds in terms of a user defined vocabulary of sound descriptors (for example, by using words in English) rather than in terms of numerical streams and (b) provides intelligent exploratory aid, so that the computer can work co-operatively with the user by offering useful mechanisms for the exploration of the capabilities of the synthesis algorithm at hand.

It is worth emphasising that we do not aim at a system tied to a specific synthesis algorithm or tied to a specific vocabulary for sound description. Rather, we aim at a software that allows the user to configure it according to the synthesis algorithm he or she wishes to use and according to a personal vocabulary for sound description.

Nevertheless, we carried out this investigation by means of the implementation of a case study system that uses a specific sound synthesis method (namely, subtractive synthesis of formants) and a specific vocabulary for sound description, which suits this synthesis method. However, throughout the investigation we addressed several issues that give us various insights towards an intelligent sound design software of

---

* To appear in a forthcoming issue of the Computer Music Journal, The MIT Press.

larger scope. Thus, in this paper we propose an AI approach to sound design systems generally, by means of a case study.

The proposed AI approach focuses sound design as a knowledge-based kind of intelligent behaviour. Thus, we consider that sound design involves the explicit organisation, application, and generation of knowledge. AI is aimed here at helping the composer to handle this knowledge by means of suitable knowledge representation and machine learning techniques.

It is worth mentioning that there have been some attempts towards the design of intelligent systems, not specifically for sound synthesis, but for sound editing. The most successful ones function as interfaces for systems that perform tasks involving audio recording studio techniques, such as mixing, multitracking, and filtering (for example, CIMS (Schmidt 1987) and Elthar (Garton 1989)). Although not primarily designed for sound synthesis, these systems in some way inspired our research in many aspects. More recently, Russ Ethington and Bill Punch published in this journal (1994) a paper about a software called SeaWave. SeaWave is an additive synthesizer in which sounds can be produced by means of a vocabulary of descriptive terms. Although of a limited scope (that is, it works only with a circumscribed vocabulary for additive synthesis provided by its programmers), SeaWave offers an excellent insight into the problem.

We begin the paper by presenting the background philosophical concepts behind our research work. Then we put forward some desirable capabilities we think are important in an intelligent system for sound design (ISSD). Next, we examine how our AI approach works. Then we propose a system architecture, followed by a functioning example. Finally, we attempt to evaluate our case study system and present some conclusions and further work. Before we begin, we would like to observe that we do not claim that low level parameter settings, or any other means of parametrical communication with a computer implemented synthesizer (for example, graphic interfaces), should be ignored. In this paper we intend to propose a novel paradigm, which we think can enhance currently available sound design systems.

Understanding Sound Design as Knowledge-based Intelligent Behavior: a Research Method

Producing a desired sound on a musical instrument, a clarinet for example, fundamentally depends upon sub-cognitive physical skills, such as operating keys, controlling the air flow rate and pressure profiles. By training and practice, musicians develop an "inner ear" with which they mentally "hear" a sound. They also develop the fine physical control required to generate and control the desired sounds on their particular instrument. This process is not typically cognitively accessible, and is therefore difficult to make explicit.

A similar situation exists for the relationship between an imagined sound, a real sound, and a sound description. If, for example, we ask our clarinettist to play a melancholy sound she would have no difficulty imagining a suitable sound and producing one that satisfies our request. However to ask her how she does this would be a much more difficult request.

If instead our clarinettist turns to computer sound synthesis, this lack of an explicit understanding of how imagined sounds are produced and described presents significant problems. The computer synthesis of sounds is fundamentally controlled, not by sub-cognitive physical skills, but by setting values in algorithms, which control

digital sound synthesis devices (for example, oscillators and filters). The task of sound design using computer sound synthesis techniques is therefore qualitatively different from sound design using acoustic instruments.

When designing computer synthesized sounds to be used in a piece of music (for example, a piece of electroacoustic music), composers have an intuition about how these sounds will be organized into a musical structure. In order to design these sounds, composers often explore a variety of possible solutions by trying out possibilities within a certain personal style or idiom.

However, to devise a suitable sound design system is not an easy task. Design is a very complex kind of intelligent behavior. It involves engaging in cognitive and physical acts in order to establish the suitability and effectiveness of creations prior to actually constructing them. Design problems are interesting here because part of the definition of the problem is given in the form of requirements the designed artefact must meet. In attempting to solve such problems, designers explore the space of possible solutions by trying out possibilities and by investigating their consequences.

One cannot hope fully to understand design by adopting a unilateral view of its study but only from the combined efforts of many different disciplines. Nevertheless, we are interested in studying only a limited aspect of design: *design as an explicitly knowledge-based kind of intelligent behavior.* Therefore we assume that it involves the explicit organization, application, and generation of knowledge.

In order to study design with this approach we need a methodology for describing and expressing aspects of the behavior being investigated, how we think this behavior can be engendered, and how we think it can possibly be aided, or even simulated by a computer system. The methodology we devised approaches this problem at three different levels: the *knowledge level,* the *symbolic level*, and the *engineering level* (Smithers, Conkie, Doheny, Logan, and Millington 1989).

To start with, the knowledge level involves the definition of certain aspects believed to be important for developing and expressing our understanding of how knowledge can be organized, used, and generated in a system. Knowledge is characterized here entirely functionally (that is, in terms of what it does) and not structurally (that is, in terms of physical objects with particular properties and relations). At this level we ought to identify what sort of knowledge a system has to possess in order to attain its goals (Newell 1991).

Regarding the knowledge level as a separate computer system level is useful as a point of departure for studying the problem, and moreover, it helps us to anticipate the kind of behavior we wish the system to display. Furthermore, the knowledge level facilitates the definition and the interpretation of the abstract structures that will be embedded in the system.

As for the symbolic level, it involves the provision of suitable data structures that hold and connect knowledge, plus the algorithms that extract and use the knowledge they contain. In our case, a knowledge representation scheme and an inference mechanism will be proposed together with techniques for automatic knowledge acquisition and concept formation. The symbolic level thus involves building a system architecture that embodies the knowledge level.

Finally, the engineering level involves the implementation of the architecture defined in the symbolic level.

In this paper we focus only on the knowledge and on the symbolic levels of our case study system. But before we study them, we introduce the desirable capabilities we think are important in an ISSD.

The Desirable Capabilities of an Intelligent System for Sound Design

By an intelligent system we mean a system that works co-operatively with the user, providing useful levels of automated reasoning in order to support laborious and tedious tasks (such as working out an appropriate stream of synthesis parameters for each desired single sound), and to aid the user in exploring possible alternatives when designing a certain sound. The desirable capabilities of such a system are discussed below.

Response to Intuitive Sound Descriptions

The specification of the synthesis parameters for producing a desired sound is a tedious and time consuming task. The composer has to input large streams of numerical values for each single sound (for example, Csound score files (Vercoe, 1991)). It is therefore desirable to communicate with the system in terms as close as possible to an intuitive, perceptually-oriented vocabulary, as opposed to specifying quantitative numerical values and low-level computer programming. We expect a system that allows the musician to compose sounds in terms of a personal vocabulary of sound descriptors (for example, by using words in English).

User Configuration

If a system is to allow communication in terms of the user's own vocabulary of sound descriptors, then this system should also be able to be configured, or modelled, according to certain basic premises ranging from the user's familiarity with the sound world to be explored, to his or her own terms for describing sounds.

Intelligent Exploratory Aid

The manipulative power of the computer is aimed here at supporting the intuition of the musician. The role of an ISSD is to aid the user to explore possible alternatives when designing a sound. Instead of an autonomous sound-generating system, we aim for a system that encourages collaboration between the user and the computer through an appropriate exploratory aid.

Ability to Learn from User Interaction

Philosophical considerations aside, it is generally recognized that the ability to learn is a prerequisite for any form of intelligence (Carbonell 1990; Suppes and Crangle 1990).

An ISSD is expected to be able to assist the user in concept formation, that is, to make generalizations, or classification rules, out of its own knowledge base or from external training data given by the user.

It is also desirable that the system is able to automatically update its knowledge in order to cope with new sorts of requirements. The system is to start with some basic knowledge about how to synthesize certain sounds and then to learn about other sounds through user interaction.

The Knowledge Level

The Body of Knowledge

The body of knowledge comprises: (a) the sound synthesis parameters, (b) the vocabulary for sound description, and (c) the goal of the system. In the following paragraphs, we study how to define suitable synthesis parameters. Next, we propose a method for defining sound descriptors. Finally, we introduce the goal of the system.

Defining the Boundaries of Abstraction of an Instrument whose Timbre does not Exist

Quite often, traditional Western music literature subdivides the process of producing sounds for a piece of music into work with discrete musical elements, such as notes and duration of notes. Musicians are then induced to think of producing sounds as a multidimensional control of these elements. Each musical element is referred to by symbols that the composer uses to notate the sounds in a score, and sometimes to think about them (for example, Iannis Xenakis's symbolic composition, in his book *Formalised Music*, Chapter VI (1963; 1971; 1992)). The performer is then requested to interpret the score by mapping these symbols into gestures towards a musical instrument.

We say that Western music traditionally has a certain boundary of abstraction which facilitates its representation (for example, the abstraction of a sound event as a note). Ignoring the inner acoustical features that produce a sound (for example, amplitude of partials and harmonic content), performers and composers tend to learn only how to obtain the desired results (which often go beyond what is represented in a score, for example, timbre quality) by acting on the control mechanism of the instrument: they learn what actions to perform from symbols arranged in a score, in order to play the music. (Actions and symbols can be regarded here as components of the body of musical knowledge.) As far as timbre, or sound quality, is concerned, we say that in the traditional Western music system, timbre is below the symbolic level of abstraction because there are no symbols for the definition of its characteristics. In this case, performers build a conceptual model of the instrument from a certain boundary of abstraction that gives very little room for significant manipulation of timbre.

Therefore, in order to think of an ISSD, one must define suitable boundaries of abstraction upon which the body of knowledge will be based. As happens in traditional Western music, when a performer maps musical symbols into gestures towards an instrument, so the computer operates with sound synthesis parameters in order to produce a sound (for example, the amplitude and frequency values for an oscillator). Thus, the definition of the device or mechanism which produces the sound - that is, the instrument - plays an important role here.

Pierre Schaeffer, in *Book I* of *Traité des objets musicaux* (Schaeffer 1966), proposes a generic notion of instrument, which we think is worthwhile taking into account when defining such boundaries. On the whole, Pierre Schaeffer proposes that any

mechanism which allows one to obtain a varied collection of sonic objects while keeping the permanence of a cause present in one's mind, is a musical instrument.

The *"permanence of a cause present in one's mind"* defines the identity of the instrument. A sound structure functions by the variation of certain aspects from one sound to another, and this variation is rendered perceptible by the permanence of certain other aspects. Michel Chion (1983) interprets Pierre Schaeffer's idea by indentifying those aspects whose variation is pertinent and form the abstract musical discourse as *values*, and those which guarantee the concrete permanence as *characters.* The law of functioning of sound structures therefore is: permanence of characters, and variation of values.

The problem with computers is that there is an excess of possibilities to play with. It is an "instrument" whose timbre does not exist until a programmer defines its boundaries and constraints. We assert that the definition of boundaries of abstraction and constraints involves the definition of a synthesis criterion which preserves certain common features (character) but allows differences to emerge (values).

In summary, this section is telling us that the definition of a suitable sound production model is very important, because it gives us a certain boundary of abstraction and constraints which enables us to define a coherent body of knowledge (for example, symbols and actions) about producing sounds.

Systematically Describing Sounds by Means of Their Attributes

There have been several studies concerning the definition of a framework to systematically describe sounds by means of their attributes (von Bismark 1971; 1974a; 1974b; Cogan 1984; Giomi and Ligabue 1992; Terhardt 1974; Slawson, 1985; 1987 to cite but a few). They are derived mainly from works in the fields of psychoacoustics and musical analysis. As it is not our aim to survey all these, we select only one example for discussion here. This example is based upon a well known model of sound production: namely, the *source/filter model.* The source/filter model is illustrated in Figure 1.

The source/filter model asserts that the characteristic of a sound is determined by its spectrum envelope's pattern. This pattern is composed of multiple "hills" called formants (see Figure 2) and it is thought of as the result of a complex filter through which a source sound passes. Each formant has a center frequency peak and a bandwidth. According to this model, the lowest two formants are the most significant determinants of sound quality.

According to Wayne Slawson (Slawson 1985; 1987), four perceptual attributes, namely *openness*, *acuteness*, *smallness*, and *laxness*, can be specified here as categories of equal-values contours in a two-dimensional space whose axes are the first (**f1**) and the second (**f2**) centre formant frequencies.

The attribute openness varies with **f1**, acuteness with **f2**, smallness with both **f1** and **f2** and laxness varies towards a "neutral" position in the middle of the space (Figure 3).

Figure 1: The source/filter model.

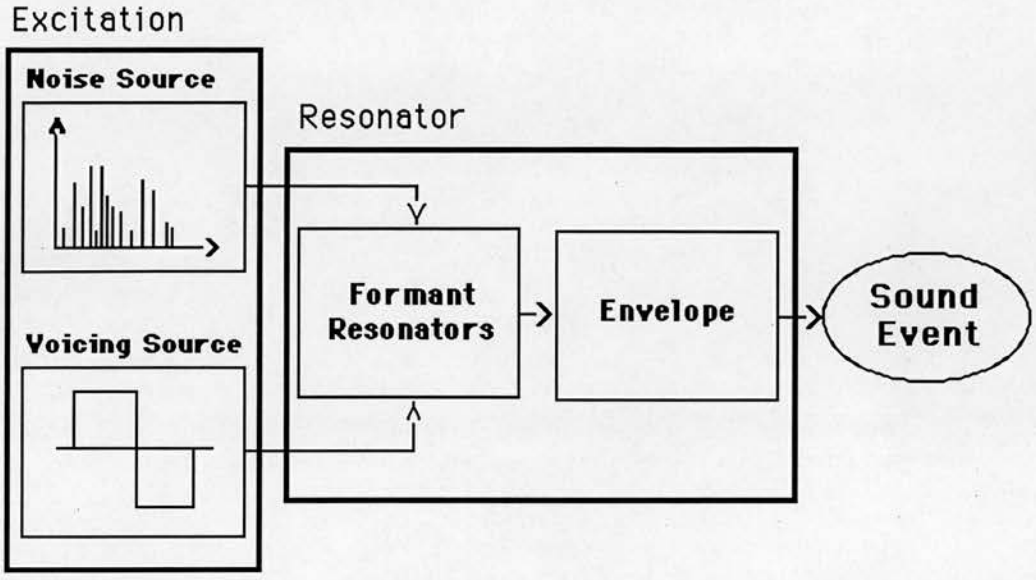
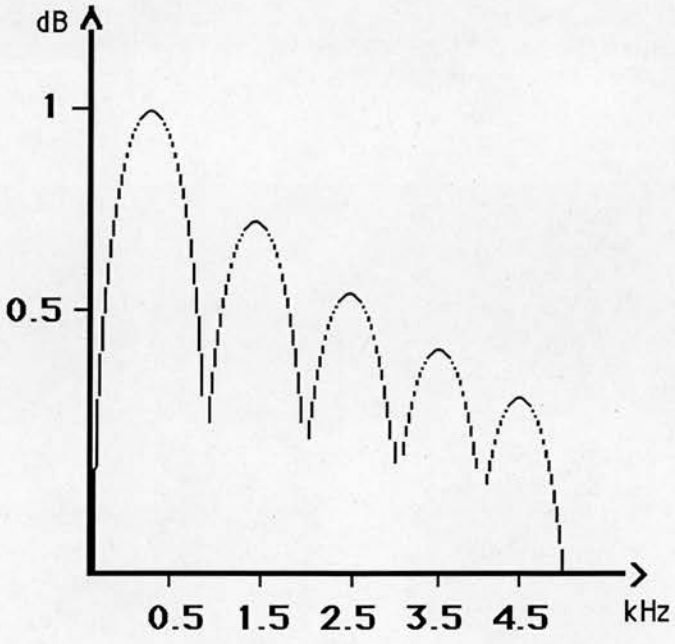


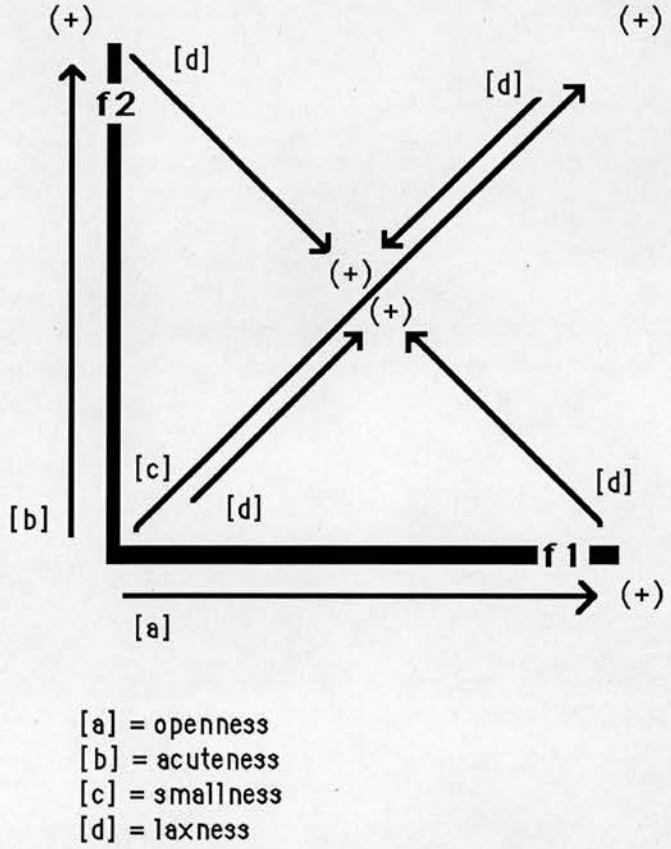
Figure 2: The frequency domain representation of the spectrum envelope's pattern of a vowel-like sound.



As for the definition of a vocabulary of sound attributes and attribute values, it is always useful to think in terms of vectors. (Note that we distinguish between two terms for describing sounds here: *attribute* and *attribute values*.) Taking as an example the attribute *openness*, we say that it corresponds to the vector given by the coordinate **f1**. The higher the value of **f1**, the higher the *openness* of the sound. After identifying a vector, the next step is to divide it into discrete sub-vectors so that each of them corresponds to different perceptual stages (or attribute values). One could define for the attribute *openness*, for example, three perceptual stages, each corresponding to a

certain first formant range of values: *low openness, medium openness* and *high openness*.

Figure 3: *Slawson's two-dimensional sound space.*



Identifying the Goal

Broadly speaking, the main difficulty in the knowledge level is the mapping between an intuitive sound description and the parametric control of computer sound synthesis. This problem is twofold.

On the one hand, apart from debates ranging from innate (biological) predisposition (Spender 1980) and natural morphology (Petitot 1989; Wishart 1985) to anthropological archetypes (Bayle 1993) and *"zoomusicologie"* (Mâche 1991); most aspects involved in sound perception, mental representation, and description, depend upon several cultural factors like personal taste, training, expertise, language, and so on. Trained listeners, for example, have a greater awareness of the mental structures they are using, implying that they have a more extensive vocabulary of description which enhances memory capacity. We have learned from the early stages of computer music (from works such as Stephen Holtzman's non-standard synthesis (1978), SSP and ASP (Berg 1975; 1980), and Iannis Xenakis' stochastic sounds, (1963; 1971; 1992)) that it is not enough to employ solely abstract mathematical models as formulae for generating sounds for a piece of music without accounting that our ear is culturally shaped.

However, there are constraints imposed by the sound synthesis technique at hand that limit the scope of the sound world. One is not supposed to ask a musician, for example, to produce a long smoothed sound with large vibrato on a harpsichord.

Hence, it does not make much sense to design an idiosyncratic system that understands a very particular vocabulary and synthesis technique. Rather than providing a system restricted to a certain particular sound world, we ought to provide means for customization.

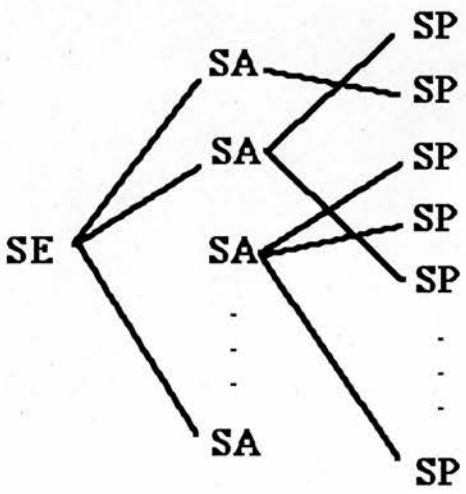
Furthermore, the ultimate goal of the system is to produce a sound from a user defined attributional description. The main problem is the mapping; the system has to know how to do it. In order to do this mapping the system has to know on the one hand, the sound synthesis parameters and, on the other hand, the vocabulary for sound description.

The Model for Action

In this section we attempt to devise a model for action for our system. The model for action ought to establish the correlation between a vocabulary for sound description and the synthesis parameters of the instrument. In other words, given a sound description, the system should work out the parameter values needed to produce this sound.

Our proposed model for action is represented in Figure 4. A sound event **SE** is composed of several sound attributes **SA**. Each **SA** in turn corresponds to one or a set of synthesis parameters **SP**. An action engine is responsible for computing the necessary **SP** values for producing a **SE**. The input for the action engine can be either the name of the desired **SE** or a set of **SA** values. Alternatively, a set of **SP** values or a set containing both **SA** and **SP** values can also be input. The output is a set of **SP** values which in turn are used for synthesizing the required **SE**.

Figure 4: *The proposed model for action.*



The action engine has to be able to compute all the necessary **SP** values for synthesizing a **SE**. If the input requirement is ill-defined (for example, an incomplete list of **SA** values), then the action engine has to be able to deduce the missing information by making analogies with other **SE**s which have similar constituents. In order to do so, the action engine has to be able to automatically *induce* rules about "prominent" sound features that will identify which **SA**s are more important for describing **SE**s. Furthermore, given that **SP** values can also be given as part of the requirement, the action engine has to be able to handle values which may not match with any known **SA**. Likewise the action engine has to be able to add this new information to the system's body of knowledge automatically .

As an example, suppose that the system has the following information in the body of knowledge:

SE = { vowel(i),
vowel(o) }

SA = { openness = { low, high },
acuteness = { low, high },
fundamental frequency = { low, medium, high } }

SP = { f1 = { 290 Hz, 400 Hz, 650 Hz },
f2 = {1028 Hz, 1700 Hz, 1870 Hz},
f0 = { 220 Hz, 440 Hz , 880 Hz} }

Symbols within braces mean possible values for the element on the left side of the equality. For example, a sound attribute **SA** may value **openness**, **acuteness**, or **fundamental frequency**. The sound attribute **openness** in turn may value either **low** or **high**, and so forth.

The action engine has to be able to infer that, for example, **vowel(i)** is described by having **low openness**, **high acuteness**, and **low fundamental frequency**. Furthermore, it has to infer that **low openness** and **high acuteness** mean **f1 = 290 Hz** and **f2 = 1870 Hz**, respectively, and that **low fundamental frequency** means **f0 = 220 Hz**.

By an "ill-defined requirement" we mean to require the synthesis of a sound - **vowel(i)**, for example - by inputting only the information **medium fundamental frequency**, for instance. In this case, the action engine has to size up the missing information needed for producing **vowel(i)**.

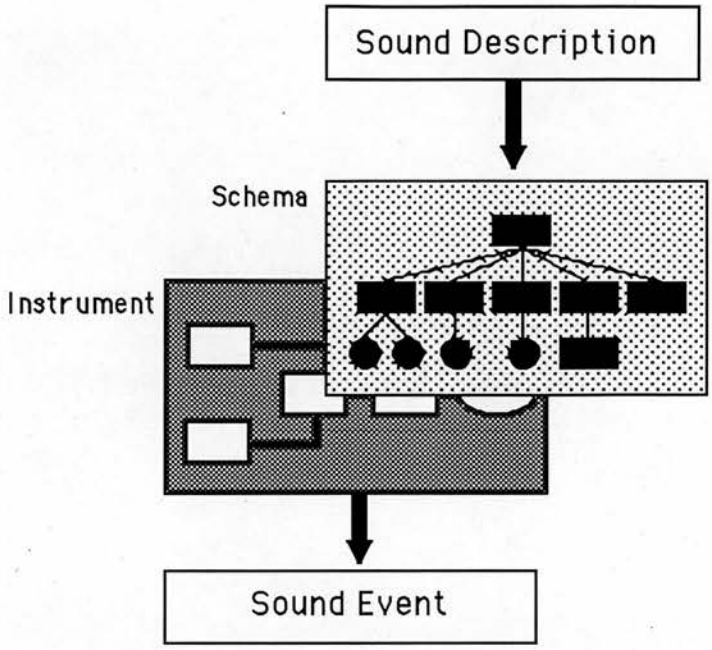
The Symbolic Level

Introductory Concepts

The Internal Representation Hypothesis and the Concept of Schemata

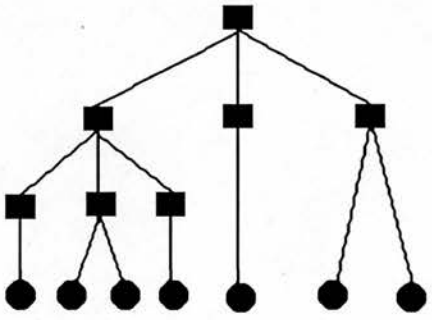
Our primary presumption, borrowed from the cognitive approach to psychology and symbolic AI, is that mental activity is mediated by internal representations. Although there is no consensus about what these representations are - images, symbols, neurophysiological states - there are a few aspects that are reasonably well explained.

Figure 5: *A schema is a multi-levelled structure aimed at mediating higher-level sound descriptions and their respective synthesis parameters.*



Stephen McAdams (1987, pp. 18), for example, indicates that the form of these representations is very important in that *"different forms of representation, even though they carry the same meaning, may have different properties with respect to what the process of transformation can do with them"*. And more, *"certain aspects of reality are more apparent in some sort of representation than in others"*.

Figure 6: *The slots of the ASS are represented as circles and grouping nodes as rectangles. A line represents a link.*



Symbolic Machine Learning

Several algorithms for symbolic learning are employed in AI systems, ranging from *learning by being told* to *learning by discovery* (Bratko 1990; Carbonell 1990). In the former case, the learner is explicitly told by a teacher what is to be learned. There is no teacher involved in the latter case. In learning by discovery, the system autonomously discovers new concepts merely from observations or by planning and performing experiments in the domain. Between these two extremes lie many other techniques.

Unfortunately, there is no ideal machine learning technique to deal with all dimensions of a domain. The criteria for selecting a machine learning technique depends upon its purposes.

As we are interested in a system that, on the one hand, is able to make generalisations and, on the other hand, is able to update its knowledge base. For this work we selected two techniques: *inductive learning* and *automatic knowledge acquisition*. Both are well known techniques that have been satisfactorily used in several expert systems (Dietterich and Michalski 1981; Quinlan 1982; Winston 1984).

The Abstract Sound Scheme (ASS)

The ASS is the abstract representation scheme we devised for representing sounds. In fact, the role of the ASS is twofold. On the one hand, it embodies a multi-levelled representation of the signal processing of an instrument. On the other hand, it provides an abstraction for representing sounds. Thus, the ASS enables the organization of knowledge about sounds based upon the signal processing model that produces them. A sound event is represented here in terms of the various perceptual components that contribute to its identity. However, these components have to be in some way tied to the signal processing model.

The ASS is constituted of *nodes, slots,* and *links*. Nodes and slots are components, and the links correspond to the relations between them. Both components and links are labelled. Slots are grouped bottom up into higher level nodes, which in turn are grouped into higher level nodes, and so forth, up to the top node (Figure 6).

The ASS is, in fact, a tree-like abstract data structure whose ultimate nodes (the leaves) are slots. Each slot has a name and accommodates either: (a) a sound synthesis datum or (b) a pointer to a procedure for calculating a sound synthesis datum.

The Notion of Sound Assemblage and Sound Inheritance

As we mentioned earlier, the ASS provides an abstract data structure for representing sound events. An instantiation of the ASS corresponds to a sound. In order to instantiate a certain sound, the ASS's slots must be "filled" with synthesis parameter values. We say that an instantiation of a sound event is an *assemblage*. For each different sound produced by the instrument there is a corresponding assemblage. Considering the case of an instrument that intends to simulate the vocal tract mechanism, for example, an assemblage would correspond to a certain position of the vocal tract producing a certain steady sound, such as an open vowel. Eventually, an assemblage might also be made to correspond to a sound which changes its "color" during its production, such as a diphthong.

Figure 7: *The assembler engine firstly collects the appropriate slot values in order to assemble the desired sound and then activates the synthesis algorithm.*

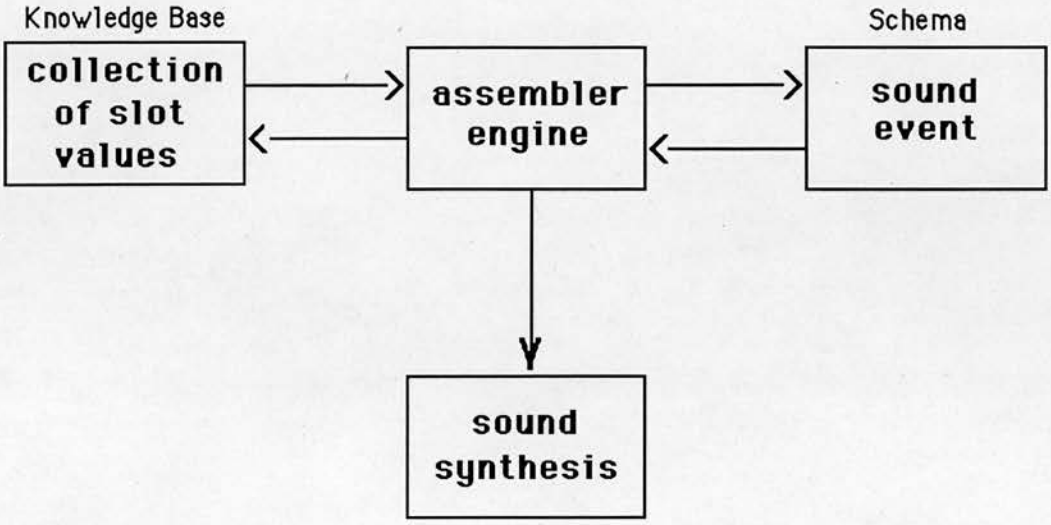
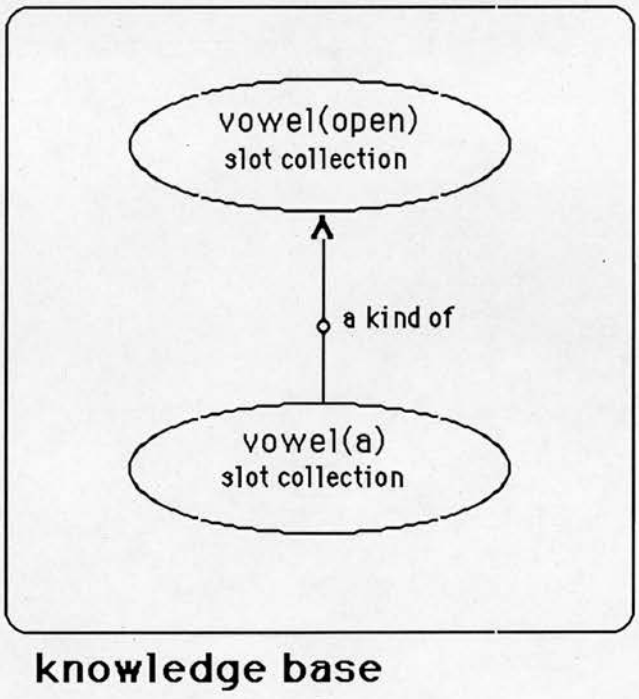


Figure 8: *An example knowledge base.*

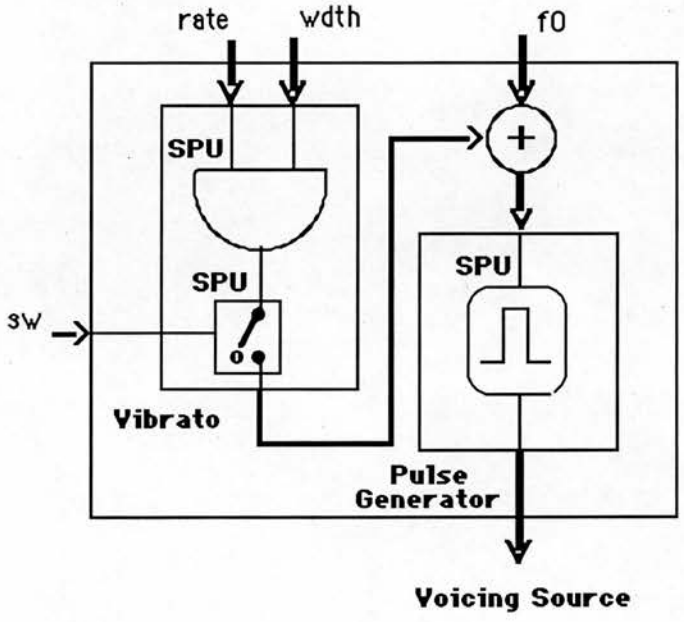


In practice, the information for instantiating the ASS is recorded in a knowledge base as a collection of slot values. This is to say that the information for the assemblage of a particular sound event is clustered around a collection of slot values. An *assembler engine* is then responsible for (a) collecting the appropriate slot values, (b) assembling the ASS and (c) activating the synthesis algorithm (that is, the instrument) in order to produce the sound (Figure 7).

Sounds, or collection of slots, are recorded hierarchically in the knowledge base. The ability to represent hierarchically the relationship between slot collections is useful for the inheritance relation. Inheritance is a mechanism by which an individual assumes the properties of its class and by which properties of a class are passed on to its subclass (Luger and Stubblefield 1989). This hierarchical organisation is accomplished by means of a link called *a kind of* (Figure 8). When a slot collection for a sound is related, by means of the link *a kind of*, to another slot collection at a higher level, the former inherits properties of the latter. Note in Figure 8 that **vowel(a)** is said to be *a kind of* **vowel(open)**. This means that slots which eventually may not be defined for **vowel(a)** will be instantiated with slot values taken from **vowel(open)**. In practice, the assembler engine has to "know" that the missing slots in one level are inherited from a higher level. Inheritance reduces the size of the knowledge base by promoting the definition of common properties only once, it helps prevent inconsistencies and redundancies in updates, and it stimulates the integration of knowledge. Besides all that, it provides means for organizing and representing sounds into a taxonomy.

As a matter of fact, one can think of partial assemblages too, that is, assemblages of single ASS nodes. This is to say that sound attributes are also represented as a collection of slots in the knowledge base. Therefore, we can have various instantiations of an attribute (that is, partial assemblages) in the same way we have them for sounds (that is, whole assemblages). Having said this, let us now study how we can ask the system to produce a sound from a qualitative description (that is, from a set of attribute=value tuples).

Figure 9: *The voicing source module of the model shown in Figure 1.*

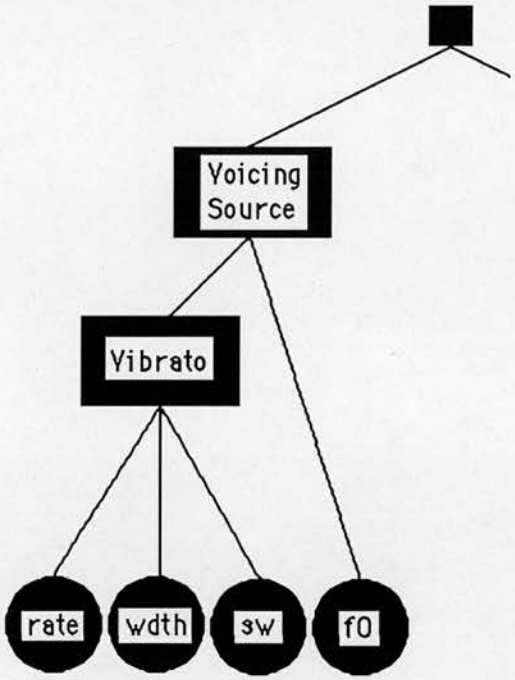


Referring to sounds by means of a user defined vocabulary of sound descriptors

For each node of the ASS one can define a set of possible instantiations, or attribute values. Let us take as an example a partial schema which represents only the *voicing source* module of a certain synthesis model (Figures 9 and 10).

This module is composed of a *vibrato* sub-module and of a slot called **f0** (for fundamental frequency). The *vibrato* sub-block, in turn, is composed of three slots: **rate** (of vibrato), **wdth** (width of vibrato), and **sw** (a switch for turning on/off the vibrato effect). David Jaffe (1994) suggests elsewhere the notion of *synthesis meta-parameter*. In our case, one could think of a node as a kind of meta-parameter.

Figure 10: *The ASS representation of the voicing source module.*



One could establish that the possible attribute values for **vibrato** are **none, uniform,** and **too slow**, for example. Each of these attribute values will then correspond to either a set of numerical values or to a set of ranges of values within a certain interval. For example, one could define that **vibrato** is **none** if **rate = 0 Hz, wdth = 0 %** of **f0**, and **sw = 0.** The node **voicing source** is similarly defined: one could establish that **voicing source** is **steady low** if **vibrato = none** and **f0 = 55 Hz**, for example. This information is also represented in the knowledge base.

Hypothetically considering only this left part of the example schema, one could request the system to produce, for example, a sound by inputting the description *steady low voicing source and no vibrato*, instead of inputting the name of the sound.

The Role of Machine Learning

The target of inductive learning here is to induce general concept descriptions of sounds from a set of examples. A further aim is to allow the machine to use automatically induced concept descriptions in order to identify unknown sounds or possibly suggest missing attributes of an incomplete sound description (such as the one we just demonstrated above). Our main reason for inducing rules about sounds is so that the system can then aid the user to explore among alternatives to design a certain sound. Here the user would be able to ask the system to *"play something that sounds similar to a vowel /a/"* or even *"play a kind of dull sound"*, for example. In these cases the system will consult induced rules for inferring which attributes are relevant for synthesizing a vowel /a/-like sound or a sound with dull color attribute.

An example rule, when looking for a description for, say **vowel(a)**, on the basis of some examples could be as follows:

A sound is **vowel(a)** if:
it has **fast vibrato** and
high openness.

No matter how many attributes **vowel(a)** had in the training set, according to the above rule, the most relevant attributes for this sound are **vibrato = normal** and **openness = high**. *"Most relevant"* here means what is most important for distinguishing **vowel(a)** from other sounds of the input training set. In this case, if the system is asked to synthesize a sound with **fast vibrato** and **high openness**, then it will play **vowel(a)** instead of synthesizing a sound from scratch. Eventually, the user could input an "incomplete" requirement, such as **low voicing source** only. In this case, the system would size up the missing information in order to assemble the ASS.

The target of automatic knowledge acquisition in our case is to allow the system to update its knowledge about attribute values by user interaction. We remind the reader that the input requirement for producing a sound can contain either or both attribute values (for example, **vibrato = none**) or slot values (for example, **f(0) = 55 Hz**). The aim of supervised deductive learning is to allow the system to infer whether or not input slot values (in a requirement) match with known attribute values. If there is no matching, then the system automatically adds this yet unknown information to the knowledge base and asks the user to label this new deduced attribute value. Suppose that the system knows three values for the attribute **vibrato**:

none if { **rate = 0** Hz, **wdth = 0** % }
uniform if { **rate = 5.2** Hz, **wdth = 3** % }
too slow if { **rate = 3.6** Hz, **wdth = 3** % }

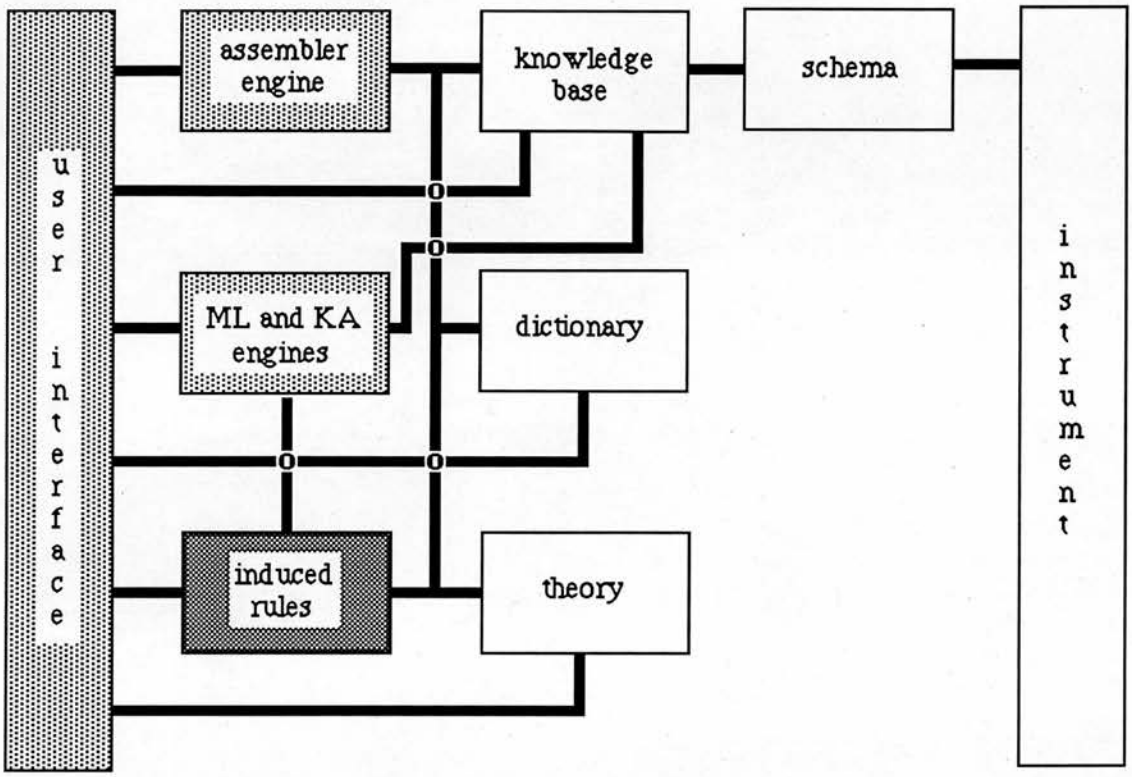
If the user requires a sound with **rate = 12 Hz**, then the system will synthesize it, and will deduce that there is no attribute value for vibrato in the knowledge base whose **rate** is equal **12 Hz**. In this case the system adds this new information to the knowledge base, works out the other slot values needed to create this new attribute value, and asks the user to label it. Let us say, for example, that the user wishes to call it **tremolo**. Eventually the system will add the following information in its knowledge base:

tremolo if { **rate = 12 Hz, wdth = 3 %** }

Towards a system architecture

User configuration is one of the desirable capabilities of an ISSD. Therefore we propose a system that features open-ended modules (Figure 11).

Figure 11: *The proposed system architecture.*



Engines and Services Provided by the System

The *assembler engine* and the *ML and KA engine* modules have already been discussed above. The *ML and KA engine* module performs two kinds of tasks: *inductive learning* and *automatic knowledge acquisition*. The training set for the inductive learning mechanism is either given by the user, or is automatically produced by making an introspection in the knowledge base. An analysis of their algorithms is beyond of the scope of this paper. They can be found in (Miranda 1994d).

The *user interface* module provides means for communicating with the system. Here the user can activate the assembler engine in order to produce a sound, consult the status of the system (such as the content of the *induced rules* module and the content of the *knowledge base* module - to be dealt with below), and input any external information the system might need (such as the names for new sounds and attributes, and training sets).

Information internally generated by the system

The *inductive rules* module holds the information internally generated by the system as the result of the inductive learning. As its name suggests this module contains rules which were induced by the *machine learning engine* module.

User specified modules

These are the open-ended modules. They define the domain of the system, that is, the sonic world the system will deal with. Here the user implements the synthesizer (both the signal processing level and the schema), the knowledge base whose information is used to "play" the synthesizer, a dictionary of slot values and a theory for the instrument.

Default libraries of such modules might be provided in case the user does not wish to start from scratch.

The instrument, the schema and the knowledge base modules

The instrument can be specified by means of any suitable SWSS (Software for Sound Synthesis) package, such as CLM (Schotstaedt 1992), Csound (Vercoe1991), Mosaïc (Morrison and Waxman 1991), SOM-A (Arcela 1994), or ISPW Max (Puckette, Lipp, and Waxman 1992), to name but a few. Having implemented the instrument then the user represents it by means of the ASS (the *schema* module).

In the *knowledge base* module the user specifies clusters of slot values. Each cluster corresponds to an instantiation of either a whole sound event or an internal node of the schema. As the system has the ability to acquire knowledge from user interaction, the user does not necessarily need to specify this information exhaustively beforehand.

The dictionary module

We mentioned earlier that the user can request the system to produce a sound by inputting (a) the name of the sound, (b) a set of attribute=value tuples, (c) a set of sound synthesis parameter values (that is, slot=values tuples), and (d) a combination of (b) and (c). Thus, it is also desirable to provide means for referring to sound synthesis parameter values by means of user defined labels (for example, words in English) as an alternative to numbers.

The meaning of these labels in terms of synthesis values is specified in the dictionary module. In the illustration we gave above when we defined the model for action, one could refer to the values of **f0** as { **low**, **high**, **medium** }, for instance. In order to do so, the following entries would have to be specified in the dictionary:

f0 = { low = 220 Hz,
medium = 440 Hz,
high = 880 Hz }

The theory module

Here the user specifies a theory for the instrument. A theory is a set of formulas for calculating slot values. These formulas can calculate values either based upon other slot values or by the random choice of a value within a certain user specified interval.

This module works as an extension of the dictionary module. This means that, instead of attaching a fixed number value to an entry of the dictionary, the user can tie the "meaning" of this entry to a formula (or rule) of the theory module. One could specify, for example, that a **low** value for the slot **f0** is any value within a certain interval between **110 Hz** and **330 Hz**, for example. Alternatively, the user can also specify a formula for calculating a slot value based on other slot values. For example, one could specify that a **medium f0** corresponds to the value of a **low f0** multiplied by two.

The ability to be able to specify a theory for the instrument is very powerful because it provides means for defining its constraints. Constraints are important here for the definition of what we referred earlier to as the "character" of the instrument. For example, one could define a constraint for our example ISSD, which states that the value of the fundamental frequency cannot exceed the value of the first formant centre frequency. (If this happens the first formant frequency will not resonate at all.)

A Brief example of Functioning

We present below a brief functioning example of our case study system. For the sake of simplicity we illustrate only the assembler engine in action. The machine learning engines will be the main subject of a forthcoming paper.

Suppose that in order to produce a sound the assembler engine would need to assemble the *voicing source* of the example shown in Figures 9 and 10. The assembler engine's action is illustrated in Figure 12, and its steps are described as follows:

(**1**) The user inputs a request: produce a **steady low sound**.

(**2**) The assembler engine finds that the description **steady low** corresponds to the attribute **voicing source** and collects the value of **f0** and **vibrato**.

(**3**) The value of **f0** is not a numerical value. In this case the assembler engine consults the dictionary. The dictionary tells that the meaning of the expression **very low** is calculated by a formula (or rule). Then the assembler engine consults the theory. The theory in turn tells that a **very low f0** is any value between **55 Hz** and **110 Hz**. Let us suppose that the assembler engine computes the value **86 Hz** for **f0**.

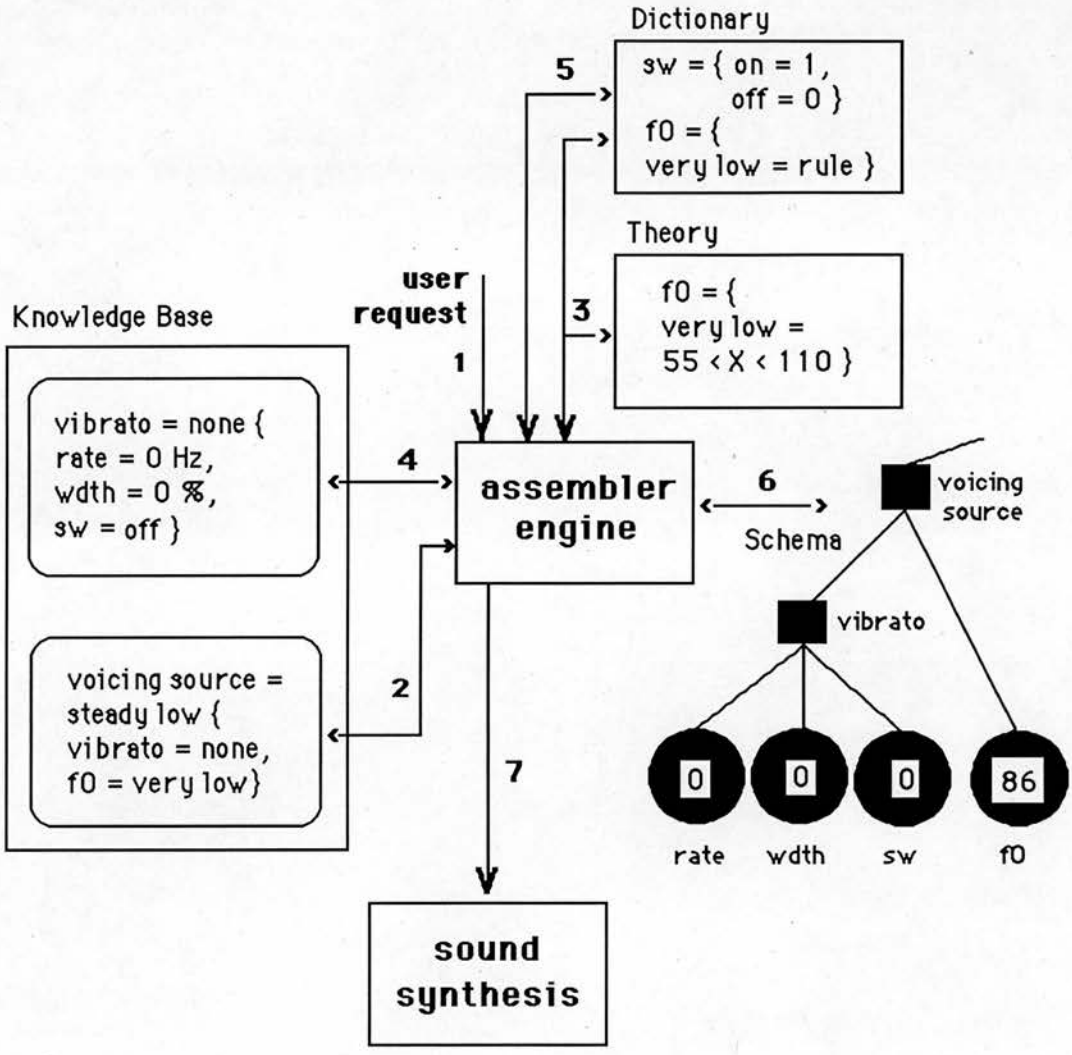
(**4**) The assembler engine finds in the knowledge base the attribute value **vibrato = none** and collects the values of the slots **rate**, **wdth**, and **sw**.

(**5**) The assembler engine consults the dictionary in order to collect the value of **sw = off**.

(**6**) The assembler engine assembles the ASS.

(**7**) Finally, the assembler engine sends the slot values to the synthesis algorithm and the sound is then synthesized.

Figure 12: *An example of the assembler engine in action.*



Evaluation

In the following paragraphs we attempt to assess this case study system by examining whether it meets the desirable capabilities mentioned earlier, at the beginning of the paper.

Response to intuitive descriptions

Does the system respond to intuitive sound descriptions? Yes, it does respond. On the condition (a) that the system is set up with a good vocabulary for attributes and slots and their values, and (b) that this vocabulary is kept coherently by the user.

User configuration

Our own criticism points to certain difficulties concerning the user configuration of such a system.

Firstly, one has to take into account that there are certain synthesis techniques that suit our approach better than others. Those techniques that allow the use of their acoustic counterpart as a metaphor to think about sounds are preferable. We cite, for example, the subtractive synthesis and the physical modelling techniques as two suitable ones. Other techniques, such as additive synthesis and distortion, possibly demand other metaphors that are beyond the scope of our research so far. This issue certainly deserves more attention in future work.

Secondly, it must be said that we spent a great amount of time and research to define a coherent vocabulary for describing the sounds which are capable of being produced by our case study instrument. Although we believe that most of this was due to the difficulty of getting a research project started properly when one is searching for a suitable point of departure, we admit that this task demands a certain erudition.

Finally, note that what is customized is the domain of the system, and not its functioning. Apart from writing certain kinds of procedures in the *theory* module, the user has no access to the algorithms of the system. Perhaps more research is needed in order to check whether the customization of certain functional aspects would enhance this sort of system.

Intelligent exploratory aid

Undoubtedly our case study system provides intelligent exploratory aid. The fact that the user can request the production of sounds by means of "incomplete" descriptions from which the system works out the necessary synthesis parameters, would be enough to attain this prerequisite. Besides this, it also features something else: the ability of inducing rules about sounds either represented in its knowledge base or given in an input training set. This means that the system not only completes the synthesis algorithm with missing values, it does it intelligently. It consults its induced rules in order to fill in this information with as much coherent data as possible.

Ability to learn

Assuredly the system has the ability to learn. It is not just able to automatically update its knowledge base, but it also has the ability to induce rules about sounds.

We would like to remind the reader that there is a limitation within the knowledge acquisition feature. The system actually does not expand its knowledge about the attributes for sound description. Rather, is acquires knowledge only about new attributes values, that is, variances, or "instantiations", of the existing attributes.

Conclusion

To start with, we can conclude that it is possible to devise a sound design system that fulfils the desirable capabilities outlined at the begining of this paper.

As for the ability to respond to qualitative sound descriptions, there are some trade-offs to be pondered. On the one hand, we have to consider the coherence of the system and, on the other hand, we have to consider its generality .

By "coherence" we mean the degree of closure amongst the vocabulary for sound description understood by the system, the synthesis algorithm that responds to this vocabulary, and the sounds produced by this algorithm in response to the vocabulary. As for "generality", we mean the flexibility of the system in the definition of the vocabulary and to the customization of the synthesis algorithm.

On the one hand, one can devise a system with a good degree of coherence by tying it to a specific synthesis technique and to a circumscribed vocabulary for sound description. In this case, all the mappings are established beforehand by its programmer. Therefore, the vocabulary will reflect the way the programmer understands a particular sound domain and it tends to be very narrow. In this case, the programmer will certainly specify everything in a very disciplined way, so that the labels of the vocabulary carry significant perceptual information to the synthesis algorithm. Thus, the degree of coherence between this vocabulary and the sounds it describes tends to be very high, but the generality of the system is kept very low.

On the other hand, one can devise a system with a good degree of generality by providing ways for the user to create a personalized vocabulary and to design his or her own synthesis algorithm. In this case, the system ought to offer a formalism for representing this information, which facilitates the specification of the mappings within it. However, the degree of coherence between this vocabulary and the sounds it is supposed to describe is not guaranteed by the system. This depends upon the skills of the person who will specify the information. In this case, the system yields a high degree of generality but does not guarantee its coherence.

As we think that a good degree of coherence can be achieved by a disciplined customization anyway, we directed our efforts towards the provision of generality.

On the whole, the strengths of our approach to ISSD design could be summarised as follows:

(**a**) We do not tie the system either to a particular sound synthesis technique or to a circumscribed vocabulary for sound description. These are entirely user defined. The link between the vocabulary for sound description and synthesis parameters is forged by the relation between the implementation of the instrument and the definition of the labels for making reference to it. It is up to the user to implement a suitable instrument so that it can support the definition of a coherent vocabulary for describing the sounds it can produce.

(**b**) The user is able to access the instrument at various levels, ranging form direct access to numerical settings to higher level descriptions of its behavior. Moreover, if the user inputs numerical settings, the system is able to deduce the possible significance of these numbers in terms of the higher level descriptions. This is done by means of an automatic knowledge acquisition engine that has the ability to update the system's knowledge about sounds and attribute values through user interaction.

(**c**) We offer powerful knowledge inference and machine learning techniques, which enhance the performance of the system with reference to exploratory assistance. Furthermore, the system also assists the user in concept formation (for example, the user is allowed to consult the rules made by the machine).

On the other hand, we can summarise the weaknesses of our approach to ISSD design as follows:

(**a**) The definition of a coherent vocabulary for sound description is not a straightforward task. We provided only a rough study, through an example very much biased to a certain class of synthesis techniques. It is still obscure how one could accomplish these independently of the synthesis model at hand.

Further Work

It is undeniable that we must find some way to build a closer link between the labels used in such a kind of ISSD and the perceptual vocabulary a composer would like to use, without losing the system's generality. Ideally, a composer should be able to depart from a perceptually-oriented vocabulary, independently of a synthesis model, and then to implement it.

Due to the fact that we began this research project assuming that the design of an ISSD essentially would involve only the modelling of certain cognitive behaviors, and not sub-cognitive ones, we have confined ourselves to a purely symbolic approach. Now that we have accomplished something, we would like to advance towards hybrid systems, by adding the support of an "analogical", sub-symbolic level to the symbolic one. We reckon that a neural network could be able to efficiently identify prominent classificatory features in input samples of (real) sounds, and also provide ways for referring to them by means of user defined labels. Neural network technology (Forrest et al. 1987) combined with the phase vocoder resynthesis technique (Wishart 1987; Dobson 1993), for example, would constitute an interesting ground for the investigation of better links between intuitive sound descriptions and synthesis parameters.

Acknowledgments

The author wishes to thank Peter Nelson (of the Music Faculty) and Alan Smaill (of the Artificial Intelligence Department) for their advice and comments on this work.

References

Arcela, A. 1994. "A Linguagem SOM-A para Síntese Aditiva." *Proceedings of the I Simpósio Brasileiro de Computação e Música.* Caxambú: Sociedade Brasileira de Computação.

Bayle, F. 1993. *musique acousmatique: propositions... ...positions.* Paris: INA/GRM & Buchet/Chastel.

Berg, P. 1975. "ASP Report." Technical report, Utrecht Institute of Sonology.

Berg, P. 1980. "SSP and Sound Description." *Computer Music Journal* 4(1): 25-35.

Bismark, G. von 1971. "Timbre of Steady Sounds: Scaling of Sharpness."*Proceedings of the 7th International Congress on Acoustics.* Budapest: Akadémiai Kiadó.

Bismark, G. von 1974a. "Timbre of Steady Sounds: A Factorial Investigation of its Verbal Attributes." *Acustica* 30:146-158.

Bismark, G. von 1974b. "Sharpness as an Attribute of the Timbre of Steady Sounds." *Acustica* 30: 159-172.

Bratko, I. 1990. *Prolog Programming for Artificial Intelligence.* Wokingham: Addison Wesley.

Carbonell, J. G. 1990. "Introduction: Paradigms for Machine Learning." *Machine Learning: Paradigms and Methods*, Carbonell, J. G. (editor), Massachusets: The MIT Press/Elsevier.

Chion, M. 1983. *Guide des objets sonore.* Paris: INA-GRM/Buchet-Chastel.

Cogan, R. 1984. *New Images of Musical Sound.* Harvard University Press.

Dietterich, T. and Michalski, R. 1981. "Inductive Learning of Structural Descriptions." *Artificial Intelligence* 16.

Dobson, R. 1993. "The Operation of the Phase Vocoder." *Composers' Desktop Project software documentation.* York: CDP.

Ethington, R. and Punch, B. 1994. "SeaWave: A System for Musical Timbre Description." *Computer Music Journal* 18(1): 30-39.

Forrest, B. M., Roweth, D., Stroud, N., Wallace, D. J., and Wilson, G. V. 1987. "Neural Networks Models." Physics Dept. preprint 87/419 (ECSP - TR - 11), University of Edinburgh.

Garton, B. 1989. "The Elthar Program." *Perspectives of New Music* 27(1): 6-41.

Giomi, F. and Ligabue, M. 1992. "Analisi Assistita al Calcolatore della Musica Contemporanea." Rapporto Interno C92-01 CNUCE/CNR, Conservatorio di Musica L. Cherubini.

Holtzman, S. 1978. "A description of an automated digital sound synthesis instrument." DAI Research Report Nr. 59, University of Edinburgh.

Jaffe, D. 1994. "An Overview of Criteria for Evaluating Synthesis and Processing Techniques." *Proceedings of the I Simpósio Brasileiro de Computação e Música.* Caxambú: Sociedade Brasileira de Computação.

Luger, G. F. and Stubblefield, W. A. 1989. *Artificial Intelligence and the Design of Expert Systems.* Redwood City: The Benjamin/Cummings Publishing Company.

Mâche, F-B. 1991, *Musique, Mythe, Nature ou les dauphins d'Arion.* Paris: Méridiens Klincksieck.

McAdams, S. 1987. "Music: A science of the mind?" *Contemporary Music Review* 2: 1-61.

Miranda, E. R., Smaill, A., and Nelson, P. 1993a. "A knowledge-based approach for the design of Intelligent Musical Instruments." *Proceedings of the X Simpósio Brasileiro de Inteligência Artificial.* Porto Alegre: SBIA/SBC.

Miranda, E. R., Smaill, A., and Nelson, P. 1993b. "A Symbolic Approach for the design of Intelligent Musical Synthesizers." *Proceedings of the X Reunion Nacional de Inteligencia Artificial.* Mexico City: Sociedade Mexicana de Inteligencia Artificial.

Miranda, E. R. 1993a. "From Symbols to Sound: AI-based investigation of Sound Synthesis." DAI Reseach Paper Nr. 640, University of Edinburgh.

Miranda, E. R. 1993b, "Modelagem do Aparelho Fonador para Síntese Digital e suas Aplicações na Música." Acústica & Vibrações 12:60-77.

Miranda, E. R. 1994a. "ARTIST: an AI-based tool for the Design of Intelligent Assistants to Sound Synthesis." *Proceedings of the I Simpósio Brasileiro de Computação e Música.* Caxambú: Sociedade Brasileira de Computação.

Miranda, E. R. 1994b. "Sound Design: an Artificial Intelligence approach." PhD Thesis, University of Edinburgh.

Morrison, J. and Waxman, D. 1991. *Mosaïc 3.0.* Paris: Ircam.

Newell, A. 1981. "The Knowledge Level." AI Magazine 1(3): 1-20.

Oliveira, M. A. O. 1981. "Kant." *Cadernos da UnB.* Brasilia: Editora UnB.

Petitot, J. 1989. "Perception, cognition and morphological objectivity." *Contemporary Music Review* 4: 171-180.

Puckette, M., Lippe, C., and Waxman, D. 1992, *ISPW Max Reference Manual,* Preliminary Release 0.17. Paris: Ircam.

Quinlan, J. R. 1982. "Semi-autonomous Acquisition of Pattern-based Knowledge." *Introductory Reading in Expert Systems,* Michie, D. (editor). London: Gordon & Breach.

Schaeffer, P. 1966. *Traité des objets musicaux.* Paris: Ed. du Seuil.

Schank, R. C. and Abelson, R. P. 1977. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures.* The Artificial Intelligence Series. Hillsdale: Lowrence Erlbaum Associates.

Schmidt, B. L. 1987. "Natural Language Interface and their application to Music Systems." *Proceedings of the 5th Audio Engineeting Society International Conference.*

Schottstaedt, W. 1992. *Common Lisp Music Documentation*, available via Internet ftp from the clm directory on the host machine ccrma-ftp.Stanford.edu.

Schottstaedt, B. 1994. "Machine Tongues XVII: CLM: Music V Meets Common Lisp." *Computer Music Journal* 18(2): 30-37.

Slawson, W. 1985. *Sound Color.* University of California Press.

Slawson, W. 1987. "Sound-color Dynamics." *Perspectives of New Music* 25(1 - 2): 156-177.

Smithers, T., Conkie, A., Doheny, J., Logan, B., and Millington, K. 1989. "Design as Intelligent Behaviour: An AI in Design Research Programme." Technical report, Dept. of AI, University of Edinburgh.

Spender, N. 1980. "Psychology of music (I-III)." *The New Grove's Dictionary of Music and Musicians*, Sadie, S. (editor), 15: 388-427.

Suppes, P. and Crangle, C. 1990. "Robots that learn: A test of Intelligence." *Revue Internationale de Philosophie* 152.

Terhardt, B. 1974. "On the Perception of Periodic Sound Fluctuation (Roughness)." *Acustica* 30: 201-213.

Vercoe, B. 1991. *Csound Manual,* available via Internet ftp from the music directory on the host machine media-lab.mit.edu.

Winston, P. 1984. *Artificial Intelligence.* London: Addison-Wesley (2nd edition).

Wishart, T. 1985. *On Sonic Art.* York: Imagineering Press.

Wishart, T. 1987. "The Phase Vocoder - Introduction and Reference Manual." *Composer's Desktop Project software documentation.* York: CDP.

Xenakis, I. 1963. "Musiques Formelles. "*La Revue Musicalle* 253-254.

Xenakis, I. 1971. *Formalized Music: Thought and Mathematics in Music Composition.* Indiana University Press.

Xenakis, I. 1992 *Formalized Music: Thought and Mathematics in Music.* New York: Pendragon Press (revised edition).